TAMPEREEN TEKNILLINEN YLIOPISTO

MIKAEL NIEMELÄ

Versionhallinta- ja tehtävänhallintajärjestelmistä saadun tiedon visualisointi ohjelmistoanalytiikan työkaluna

Diplomityö

Tarkastajat: Kari Systä ja Terhi Kilamo
Tarkastajat ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekuntaneuvoston
kokouksessa 03.09.2014

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
MIKAEL NIEMELÄ: Versionhallinta- ja tehtävänhallintajärjestelmistä saadun tiedon visualisointi ohjelmistoanalytiikan työkaluna
Diplomityö, 46 sivua, 14 liitesivua
Helmikuu 2015
Pääaine: Ohjelmistotuotanto
Tarkastajat: professori Kari Systä ja tutkijatohtori Terhi Kilamo
Avainsanat: ohjelmistoanalytiikka, visualisointi

Need for Speed -tutkimuskonsortion [18] päämääränä on edistää suomalaisten ohjelmistoalan yritysten kilpailukykyä tutkimalla reaaliaikaisia liiketoimintamalleja. Ohjelmistoanalytiikka ja dataan pohjautuva päätöksenteko ovat yksi tapa pyrkiä tähän. Ohjelmistoanalytiikan avulla pyritään paljastamaan ongelmakohtia projekteissa. Kun ne on paljastettu, niihin voidaan kehittää ratkaisuja, ja siten parantaa tuottavuutta. Esimerkiksi visualisaatioilla voidaan havainnollistaa projektin tilaa, ja siten löytää mahdollia ongelmakohtia. Visualisaatioista on enemmän hyötyä, mikäli niitä saadaan päivitettyä nopeasti, jotta ne kuvaisivat mahdollisimman ajantasaisesti projektin tilaa. Pelkästään yhden yrityksen dataa varten kehitetyt visualisoinnit eivät ole kovin uudelleenkäytettäviä, mikä hidastaa uusien visualisaatioiden tekemistä tai niiden pitämistä ajan tasalla. Tässä diplomityössä on iteratiivisesti kokeiltu ja kehitetty visualisaatioita projektin tarpeisiin, aloittaen tiettyyn tarkoitukseen tehdyistä visualisaatioista. Visualisaatioissa käytettiin tehtävän- ja versionhallintajärjestelmistä saatua dataa, ja tarkoituksena oli kokeilla mitä tällaisesta datasta tehdyistä visualisaatioista selviää. Myöhemmin alettiin kiinnittää huomiota visualisaatioiden uudelleenkäytettävyyteen, ja siksi kehitettiin D3.js -kirjaston [13] avulla visualisointikirjasto, jonka tarkoitus oli lisätä uudelleenkäytettävyyttä, ja siten helpottaa uusien visualisointien tekemistä. Tuloksena saatu versio ei täysin saavuttanut tavoitteita, sillä monet työvaiheet vaativat edelleen koodin muutoksia, tai dataa käsittelevien komentosarjojen ajamista, minkä takia työmäärä uuden visualisaation tekemiseen jäi liian suureksi. Tämän takia työn tuloksena tehty kirjasto jäi kokeiluksi, mutta sen tekemisessä opittuja asioita voidaan soveltaa seuraavan version tekemisessä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
MIKAEL NIEMELÄ : Visualization of version control and issue tracking data as a tool of software analytics
Master of Science Thesis, 46 pages, 14 Appendix pages
February 2015
Major: Software Engineering
Examiner: professor Kari Systä and postdoctoral researcher Terhi Kilamo
Keywords: software analytics, visualization

Need for Speed research consortium [18] aims to increase competitiveness of Finnish software companies by researching real time business models. Software analytics and data based decision making are one way to accomplish this. With software analytics it is possible to reveal hidden problem in projects. When they have been revealed, solutions can be developed to said problems, and as a consequence productivity will be improved. For example, it is possible to graphically visualize state of a project to find problems in it. Visualizations are more useful if they can be updated rapidly, so that they would depict as recent state of the project as possible. Visualizations made just for the purposes of analyzing data from one company aren't particularly reusable, which makes it slower to make new visualizations or to keep them up to date. In this thesis we have iteratively tried visualizations for needs of the project, starting from visualizations made for a single purpose. We used data from version control systems and issue tracking systems, with the purpose to examine what can be found out from such data. Later we put more emphasis on reusability, and because of that we developed a reusable visualization library with D3.js library [13]. The version presented in this thesis didn't fully reach its goals in reusability, as too many phases in work still required manually changing the code or running scripts that parse the data. Because of these limitations the library is mainly useful as a proof of concept, but lessons learned during its making can be used when developing next version of the tool.

ALKUSANAT

Diplomityö toteutettiin työsuhteessa Tampereen teknilliselle yliopistolle. Työ tehtiin osana Need for Speed -projektia, joka pyrkii parantamaan suomalaisten ohjelmistoalan yritysten kilpailukykyä.

Haluan kiittää Tampereen teknillistä yliopistoa ja työn tarkastajia Kari Systää ja Terhi Kilamoa saamastani tuesta ja ohjauksesta sekä mahdollisuudesta sovittaa diplomityön kirjoittaminen yhteen töiden kanssa. Kiitos kuuluu myös muille projektiin osallistuneille ihmisille, kuten Anna-Liisa Mattilalle, joka suunnitteli visualisaatioita, ja Aleksi Kelloniemelle, joka oli mukana tekemässä diplomityössä kuvattua työkalua.

Lisäksi haluan kiittää vanhempiani ja ystäviäni tuesta ja sopivasta painostuksesta opintojen eteenpäin viemiseksi.

Tampereella 20.02.2015

Mikael Niemelä

mikael@modeemi.fi

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

API	Ohjelman rajapinta, jonka avulla ohjelmaa voi käyttää sen ulkopuolelta. (engl. application programming interface)
CSS	Internet-sivujen ulkoasun määrittämiseen tarkoitettu kieli. (engl. Cascading Style Sheets)
CSV-tiedosto	Tapa esittää taulukkoja tekstitiedostona, arvot on erotettu esimerkiksi pilkulla tai puolipisteellä. (engl. comma separated values)
Digile	Yksi Suomen innovaatiopolitiikkaa toteuttava organisaatio, joka pyrkii tehostamaan toimialansa kehitystyötä.
DOM-elementti	Internet-sivulla oleva elementti, esimerkiksi nappi. (engl. Document Object Model)
HTML	Internet-sivujen tekemiseen käytetty kieli. (engl. Hypertext Markup Language)
IDE	Ohjelmointiympäristö, jossa on vähintään tekstiedori ja kääntäjä. (engl. integrated development environment)
koostaminen	Ohjelman koostaminen. (engl. build)
muutoksen vienti	Versionhallintajärjestelmään viety muutos. (engl. commit)
MVC-malli	Ohjelmien suunnittelumalli, jossa ohjelma jaetaan ohjaimeen, malliin ja näkymään.
skeema	Tietokannan rakenne.
sprintti	Ketterissä menetelmissä usein käytetty apuväline kehityksen helpottamiseksi. Sprinttiin, joka kestää yleensä yhdestä kolmeen viikkoa, kuuluu joku määrä tehtäviä, jotka pyritään tekemään sprintin aikana.
suoritustallenne	Tallenne ohjelman tekemistä järjestelmäkutsuista. (engl. execution trace)
SVG	Vektorigrafiikan tekemiseen tarkoitettu kieli. (engl. Scalable Vector Graphics)
tietokantavedos	Tietokannan sisältö tietyltä hetkeltä, joka on tallennettuna esimerkiksi tekstitiedostoon.

1. JOHDANTO

Ohjelmistoanalytiikka analysoi ohjelmistoja ja niiden kehitysprosesseja. Se pyrkii kuvaamaan ohjelmistokehitysprosesseja, ja tuottamaan tuloksia, joilla näitä prosesseja voitaisiin parantaa ja siten lisätä tuottavuutta [30]. Tämä on erityisen tärkeää ohjelmistojen kehityksen nopeutuessa sekä koon kasvaessa, ja niiden siten muuttuessa entistä virhealttiimmiksi. Samalla on siirrytty perinteisistä menetelmistä, kuten vesiputousmallista ja raskaasta, etukäteen tehdystä ohjelmistojen tarkasta speksaamisesta, ketteriin menetelmiin, joilla pyritään lisäämään tuottavuutta ja vähentämään kehityksen kustannuksia. Ketterissä menetelmissä käytetään usein jatkuvaa integraatiota, jolla tarkoitetaan prosessia, jossa koko ohjelma koostetaan (engl. build) ja integroidaan usein, esimerkiksi päivittäin. Perinteisissä menetelmissä integrointi tehdään usein vasta projektin loppupuolella, mistä voi aiheutua ikäviä yllätyksiä, mikäli ohjelman osat eivät toimikaan yhdessä kuten oli suunniteltu. Jatkuvalla integroinnilla vältytään tällaisilta ongelmilta, ja se mahdollistaa ohjelmiston julkaisemisen hyvinkin nopeasti. Tällaisessa prosessissa on tarvetta projektin tilaa kuvaavalle, ja nopeasti päivitettävälle analytiikalle. Esimerkiksi nopeasti tehtävät tai automaattisesti päivitettävät visualisaatiot voivat auttaa kuvaamaan prosessin tilaa kyseisellä ajanhetkellä.

Tekesin rahoittama ja Digilen koordinoima Need for Speed -tutkimuskonsortio [18] pyrkii parantamaan suomalaisten ohjelmistoyritysten kilpailukykyä. Se tutkii reaaliaikaisia liiketoimintamalleja ja sillä on kolme pääasiallista painopistettä: arvon tuottaminen reaaliajassa, syvällinen asiakastuntemus ja niinsanottu “elohopeabisnes”. Arvon tuottamisella reaaliajassa tarkoitetaan tuotekehitysprosessin nopeuttamista, jotta uusia ominaisuuksia ja tuotteita voitaisiin toteuttaa nopeammin ja pienemmällä työmäärällä. Tähän pyritään mm. ketterillä menetelmillä, jatkuvalla integraatiolla, testien automatisoinnilla ja jatkuvalla uusien versioiden käyttöönotolla. Myös dataan perustuva päätöksenteko ja ohjelmistoanalytiikka on yksi työkalu tuotekehitysprosessien parantamiseen. Sen avulla pyritään paljastamaan ohjelmistoprojekteissa olevia piileviä ongelmia nopeammin kuin aiemmin. Kun ongelmat on paljastettu, niihin voidaan kehittää ratkaisuja, ja siten parantaa tuottavuutta. Datan visualisointi erilaisilla kuvaajilla on yksi tapa havainnollistaa dataa, ja siten helpottaa ongelmien paljastamista. Hyvin tehty kaavio on usein huomattavasti intuitiivisempi keino hahmottaa dataa kuin sen esittäminen esimerkiksi taulukko-

muodossa.

Tämän diplomityön aiheena on visualisointien kehittäminen Need for Speed -projektissa tehtävän data-analyysin avuksi. Projektin osana visualisoitiin tehtävänhallintaohjelmistoista, tässä tapauksessa Jirasta, saatua dataa yhdessä versionhallintajärjestelmistä CVSAnalY:n [14] avulla saadun, viedyistä muutoksista (engl. commit) kertovan datan kanssa. Tarkoituksena oli kokeilla, mitä tällaisesta datasta tehdyillä visualisaatioilla saa selville projektin tilasta. Jirasta kerätty data saatiin suoraan yhteistyöyhtiöltä, ja he myös ajoivat CVSAnalY:n ja antoivat anonymisoidun datan tietokantavedoksena (engl. SQL dump). Visualisointien tekeminen aloitettiin kokeilemalla valmiita työkaluja, kuten Google Chartsia [20], mutta ne todettiin tarkoitukseen riittämättömäksi. Tämän takia siirtyttiin käyttämään D3.js -kirjastoa [13]. Se on matalan tason kirjasto selainpohjaisten SVG- (engl. Scalable Vector Graphics) tai HTML-kaavioiden (engl. HyperText Markup Language) piirtämiseen, eikä siinä ole valmiita kaavioita, vaan ne tulee tehdä itse peruselementeistä. Tämän takia sillä voi kuitenkin tehdä lähes millaisia kaavioita tahansa, toisin kuin valmiilla työkaluilla. Tehdyn visualisaation uudelleenkäytettävyys oli kuitenkin huono, ja uuden datan lisäämisessä oli monta käsin tehtävää vaihetta. Tämän takia pyrittiin iteratiivisesti kehittämään N4S -projektin tarpeisiin sopivaa visualisointikirjastoa, jonka avulla visualisointien tekeminen esimerkiksi uuden yrityksen datasta olisi huomattavasti nopeampaa. Tämä kuitenkin osoittautui varsin haastavaksi ongelmaksi, ja tässä työssä esitelty versio vaatii vielä jatkokehitystä ollakseen tarkoitukseen sopiva.

Diplomityön luvussa 2 on lyhyt kirjallisuuskatsaus ohjelmistoanalytiikkaan ja sen historiaan, minkä lisäksi siinä käsitellään esimerkkejä ohjelmistoanalytiikan käytännön sovelluksista. Luvussa 4 käsitellään datan visualisoinnin teoriaa ja esitellään käytetty data, sekä niiden lähteet ja käytetyt työkalut. Luvussa 5 kuvataan visualisointikirjaston kehityksen kulku, sekä esitellään tuloksena syntynyt kirjasto, minkä jälkeen luvussa 6 vertaillaan tuloksena saatua kirjastoa valmiisiin työkaluihin ja kerrotaan siitä, mitä kehitettävää työkalussa on, ja millaiseen jatkokehitykseen päädyttiin. Lopussa luvussa 7 on työstä tulleet johtopäätökset ja yhteenveto.

2. OHJELMISTOANALYTIIKKA

Ohjelmistoalan muutosten ja Internetin käytön yleistymisen myötä tiedeyhteisössä on kasvamassa määrin kiinnostuttu ohjelmistoanalytiikasta, joka pyrkii analysoimaan ohjelmistoja ja niiden kehitysprosesseja. Ketterien menetelmien yleistyessä ja ohjelmistoprojekteista saadun datan määrän kasvaessa ohjelmistoanalytiikalle löytyy enemmän tarvetta ja sovelluskohteita kuin aiemmin. Tim Menziesin ja Thomas Zimmermannin artikkelissa Software Analytics: So What? [30] käsiteltiin ohjelmistoanalytiikkaa yleisellä tasolla, ja se toimi johdantona IEEE Software -lehdessä olleelle artikkelisarjalle. Mikäli ei mainita toisin, tässä luvussa viitataan kyseiseen artikkeliin.

Internetin ja avoimen lähdekoodin projektien ansiosta dataa ohjelmistoprojekteista on niin paljon, että sitä on mahdotonta selata läpi käsin. Esimerkiksi yhdessä suuressa projektissa voi olla satoja tuhansia virheraportteja, ja erilaisten projektien määrä on hyvin suuri. Datan käsittelemiseen ja analysointiin tarvitaankin jonkin asteista automaatiota. Tärkeä osa ohjelmistoanalytiikkaa on, että se antaa suuntaviivoja siitä, miten pitäisi toimia. Käytännössä toimintaohjeita tuottavan analyysin pitäisi olla saatavilla reaaliajassa, eli nopeammin kuin projektissa tapahtuu muutoksia. Päätösten pitäisi perustua nykyiseen, eikä vanhentuneeseen dataan. Tämä on oleellinen asia, sillä perinteinen datan keruu ja analysointi, esimerkiksi datan kerääminen tai käsitteleminen käsin, on monesti liian hidasta.

Suureksi osaksi ohjelmistoanalytiikassa on kyse siitä, kuinka ohjelmistoprojektit voivat oppia itsestään ja toisistaan. Kun tarkastellaan menneiltä vuosikymmeniltä olevia tutkimuksia aiheesta, löydetään väitteitä siitä, että ohjelmistoanalytiikka mahdollistaa asioiden jakamisen projektien välillä. Tällaisia asioita ovat esimerkiksi yleisten mallien, näkemysten, datan ja metodien jakaminen, kuten tekniikat joiden avulla löydetään datasta lokaaleja malleja [36], [5], [11]. Esimerkiksi Windows Vistan tapauksessa Christian Bird ja hänen kollegansa huomasivat, että laadukkaita ohjelmistoja voi tehdä hajautetuilla tiimeillä, kunhan hallinto on järjestetty koodin toiminnallisuuden eikä organisaation osien fyysisen sijoittelun perusteella [11]. Aiemmin keskityttiin yleisten mallien löytämiseen, kun taas myöhemmin on pyritty löytämään yleisiä metodeja, joilla voidaan löytää paikallisesti hyödyllisiä asioita. 40 vuotta sitten päämääränä oli siis löytää yksi oikea malli ohjelmistokehitykseen, mutta myöhemmin on hyväksytty, ettei yhdessä projektissa opittuja asioita voida ai-

na soveltaa toiseen projektiin. Esimerkiksi Internet Explorerin vikojen esiintymistä ennustava malli ei välttämättä ole sovellettavissa Firefoxiin. Tavoitteen muuttuminen johtui kahdesta asiasta, 1900-luvun lopulla saatiin käyttöön uuden sukupolven datanlouhinta-algoritmit ja avoimen lähdekoodin projekteista saatiin suuria määriä dataa. Siksi datanlouhinta eri algoritmeilla lisääntyi, ja ennenpitkää huomattiin, ettei voida tehdä yhtä kaikkia ohjelmistoprojekteja kuvaavaa globaalia mallia. Siksi on erittäin tärkeää voida tehdä nopeasti johtopäätöksiä suuresta datamäärästä, ja tehdä siitä hyödyllisiä lokaaleja malleja.

Nykyisen pilvilaskennan aikana on yleinen harhakäsitys, että suurta datamäärää voidaan analysoida vain hankkimalla suuri määrä laskentakapasiteettia ja ajamalla sillä jotain hajautettua algoritmia. Vaikka suuri määrä laskentatehoa on hyödyllistä, se ei ole kuitenkaan tärkein asia. Huomattavasti tärkeämpää on asiantuntemus, jolla laitteistoa käytetään. Toinen virheellinen näkemys on ohjelmiston rooli, jotkut ajattelevat, että oikeilla työkaluilla kaikki ongelmat ratkeavat välittömästi. Valitettavasti näin ei ole, sillä kaikissa standardoiduissa datan analysointityökaluissa on sisäänrakennettuja oletuksia, joiden takia ne sopivat tiettyihin käyttöympäristöihin. Jos ei tiedetä, mikä kyseisessä käyttöympäristössä on tärkeää, joudutaan improvisoimaan ja ottamaan selvää asioista.

Ohjelmistoanalytiikkaa voidaan myös jakaa eri kategorioihin. Yksi tapa on jakaa ne sisäiseen ja ulkoiseen analytiikkaan. Käytännössä ero on siinä, onko data vanhaa vai tämänhetkistä dataa. Sisäinen tiimi pääsee helposti käsiksi reaaliaikaiseen dataan, mutta datan siirtämisessä ulos yrityksestä on usein käytännön ongelmia, kuten tietoturvakäytännöt ja palomuurit. Lisäksi ulkoinen tiimi joutuu usein anonymisoimaan informaatiota tietoturvan vuoksi, mikä voi olla ongelma, koska se voi hukata oleellista tietoa datasta.

Toinen tärkeä tapa jaotella ohjelmistoanalytiikkaa on jakaminen kvantitatiivisiin ja kvalitatiivisiin menetelmiin. Kvantitatiivisiin menetelmiin kuuluvat mm. perinteiset datanlouhintatyökaluilla tehtävät automatisoidut tehtävät, kun taas kvalitatiiviset menetelmät, kuten kehittäjien haastattelu [10], ovat yleensä manuaalisempia ja vaativat enemmän vuorovaikutusta. On myös tärkeää ottaa huomioon kohderyhmä (esim. kehittäjät, hallinto jne.) ja se, että onko tarkoitus tutkia uusia menetelmiä, vai ottaa joku menetelmä tai työkalu käyttöön yhtiön ohjelmistokehityksen parantamiseksi.

2.1 Lähestymistavat ohjelmistoanalytiikkaan

Artikkelissa Searching under the Streetlight for Useful Software Analytics [25] käsitellään tasapainottelua helposti sovellettavan ohjelmistoanalytiikan ja tuottoisamman, mutta työläämmän ja kehittäjien yksityisyyden vaarantavan analytiikan välillä. Helposti kerättävillä ja analysoitavilla metriikoilla voi olla vain vähän vaiku-

tusta kehitykseen tai työympäristön sosiaaliseen ympäristöön. Valitettavasti, mitä helpompaa datan keräys on, ja mitä vähemmän kiistanalaista sen käyttö on, sitä rajoitetumpaa sen hyödyllisyys ja yleisyys on. Esimerkiksi tietovarastoista (engl. repository) on helppo kerätä dataa, ja koska ne ovat usein julkisia, eikä datan keräys haittaa kehittäjiä. Kuitenkin tällaisesta datasta saadut tulokset kuvaavat vain pientä osaa ohjelmiston kehitykseen liittyvästä toiminnasta. Toisaalta tällaista julkisen datan käyttämistä syvällisempään analytiikkaan pystyvät menetelmät voivat aiheuttaa kehittäjille enemmän ylimääräistä työtä, tai niillä voi olla sosiaalisia sivuvaikutuksia. Kehittäjät voivat huolestua omasta yksityisyydestään tai siitä, että hallinto käyttäisi dataa väärin, kuten esimerkiksi yt-neuvotteluiden aikana.

Humphreyn kirjassa A Discipline for Software Engineering [22] käsiteltiin muunmuassa Personal Software Processin (PSP) käyttöä, ja kirjan versio käyttää yksinkertaisia taulukkoja, sekä manuaalista datan keräystä ja analysointia. Tähän menee kuitenkin huomattavasti aikaa, esimerkiksi yhdessä versiossa kehittäjän tulee täyttää 12 kaavaketta projektisuunnitelman yhteenvedosta koodin tarkistuslistaan. Näissä kaavakkeista saadaan tyypillisesti yli 500 arvoa, jotka kehittäjien tulee laskea käsin. Kirjassa manuaalisuutta pidetään kuitenkin positiivisena asiana, sillä automatisoidulla työkalulla olisi vaikea muokata datan keräystä tiettyyn tilanteeseen sopivaksi. PSP:n manuaalisuus siis tekee siitä joustavan, ja se rohkaisee käyttäjiään etsimään parhaiten tarpeisiin sopivaa analytiikkaa. Esimerkiksi kehittäjä, joka epäilee kokemiensa keskeytysten vaikuttavan tuottavuuteen, saa PSP:stä tällaisia tilanteita varten tekniikoita, joilla voi tehdä asiasta todisteisiin pohjautuvia johtopäätöksiä suhteellisen pienellä vaivalla.

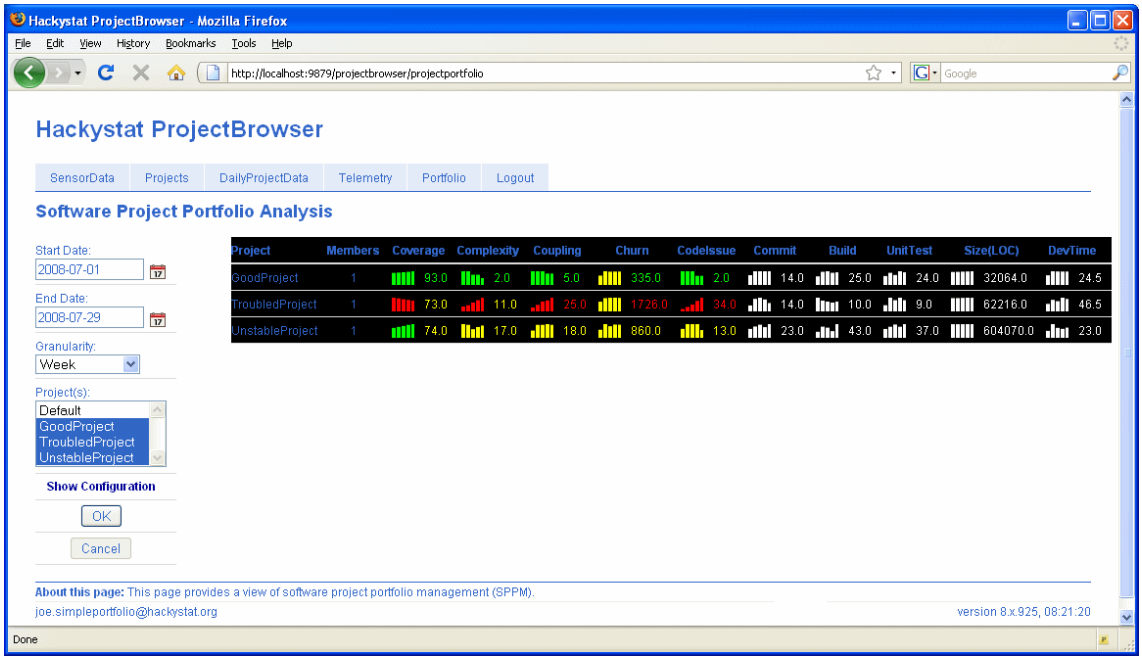
Toisaalta PSP:n manuaalisuus huonontaa saadun datan laatua. Asian tutkimisessa selvisi, että datan matalahkosta virhemäärästä (alle 5%) huolimatta ne johtivat välillä vääriin lopputuloksiin, minkä takia tutkijat kehittivät Leapin (engl. lightweight, empirical, antimeasurement dysfunction, and portable software process measurement) [24]. Leap yritti ratkaista datan laatuun liittyvät ongelmat automatisoimalla ja normalisoimalla datan analyysin. Vaikka kehittäjä antaakin suurimman osan datasta manuaalisesti, Leap automatisoi PSP:n analyysit ja joskus tuottaa analyysejä kuten regressiota, jota PSP ei anna. Leapin lähestymistapa on kevyt, koska se ei määritä kehitysprosessia toisin kuin PSP, ja se yrittää välttää kehittäjien epäluuloja antamalla kehittäjien hallita tietoja ja jättämällä kehittäjien nimet pois kerätystä datasta. Leap myös tukee kehittäjien siirtymistä projektista ja organisaatiosta toiseen säilyttäen historian.

Jälkikäteen Leapin kuitenkin huomattiin vaarantavan yhden PSP:n parhaista ominaisuuksista [25]. Automaation käyttöönottamisella Leap aiheuttaa sen, että vaikka tiettyjä analyysejä on helpompi tehdä ja datan laatu paranee, osan tekeminen vaikeutuu. Sen sijaan että kirjoittaisi taulukkoon uutta dataa, pitäisi tehdä uusi

komponentti Leapiin. Useamman vuoden jälkeen Leapin tekijät tulivat tulokseen, ettei PSP:n lähestymistapaa voitaisi koskaan täysin automatisoida, ja se väkisinkin vaatisi huomattavan määrän manuaalista datan tallennusta. Toisaalta, he olivat kehittäjien kanssa samaa mieltä siitä, että niin suuri ylimääräinen työ kehityksessä ei yleensä maksa itseään takaisin, varsinkin jos projektit eroavat merkittävästi toisistaan, jolloin historiallinen data ei ole hyödyllistä. Seuraavaksi tutkijat päättivät hylätä PSP:n tukemisen ja kokeilla millaisia tuloksia he voisivat saada, jos datan keräys ja analysointi eivät vaatisi ylimääräistä vaivaa.

Perinteisesti ensin on määritelty korkean tason tavoitteet, ja vasta sen jälkeen on mietitty datan keräystä ja analysointia. Hackystat -projektissa [26] toimittiin kuitenkin päinvastoin: ensin tutkittiin tapoja kerätä dataa ilman että kehitystiimin tarvitsee nähdä paljoa vaivaa sen eteen. Hackystatissa kehitystyökaluihin liitetyt sensorit lähettävät dataa keskitetylle palvelimelle, ja dataa kerätään myös palvelimen päässä (esim. tietovarastodata). Projektissa kehitettiinkin alusta asti mittaustyökaluja editoreille, käännös- ja testaustyökaluille. Toinen ominaisuus on huomaamaton datan keräys, kehittäjille yksi turhauttavimmista asioista on se, jos he joutuvat keskeyttämään työnsä kirjoittaakseen ylös, mitä tekivät. Lisäksi Hackystat voi kerätä dataa erittäin lyhyiltä aikaväleiltä, jopa sekunneittain, tai kun kehittäjä muokkaa metodia kooditiedostossa tai tekee testitapausta kyseiselle metodille. Hackystat osaa myös seurata ryhmätyötä, esimerkiksi sitä kuinka eri kehittäjät muokkaavat samaa tiedostoa. Projekti johti useisiin teknisiin innovaatioihin [25], kuten työkaluihin, suurteholaskennan tukemiseen, operatiiviseen määritelmään testiohjatulle kehitykselle ja ICU:un (engl. intensive care unit), joka arvioi projektin "terveyden" itsessään ja suhteessa muihin projekteihin. Kuvassa 2.1 on esimerkki Hackystatin tuottamasta analyysistä.

Vaikka Hackystatin teknologiaa on otettu mukaan jopa kaupalliseen tuotteeseen, siinä on kolme huomattavaa sosiaalista ongelmaa. Datan keräyksen huomaamattomuuden oli tarkoitus olla ominaisuus, mutta jotkut kehittäjät pitivät sitä ongelmana. He eivät halunneet asentaa instrumentointivälineitä, jotka keräävät dataa heidän toiminnastaan kertomatta siitä heille. Toiseksi, asiakasohjelman puolella tapahtuva hienojakoinen datan kerääminen voi aiheuttaa epäsopua kehittäjätiimissä. Lisäksi kehittäjät kertoivat, etteivät he pitäneet siitä, että hallinto pääsee käsiksi kehittäjien työtavoista kertovaan dataan, huolimatta lupauksista olla käyttämättä sitä väärin. Hackystatin ICU:n kyseenalaisimmat ominaisuudet ovat kehittäjäkohtaiset arviot siitä, kuinka paljon kehittäjä käyttää aikaa IDE:ssä (engl. Integrated Development Environment) työskennellen projektiin liittyvien tiedostojen kanssa ja kuinka paljon muutoksia (engl. commit) kehittäjä vie tietovarastoon ja paljonko ohjelmarivejä näissä on. Lisäksi järjestelmä kertoo kuinka usein kehittäjät koostavat (engl. build) ohjelmiston ja ovatko koosteet onnistuneita, ja sen kuinka usein kehittäjät tekevät



Kuva 2.1: Esimerkkikuva Hackystatin tuottamasta analyysistä. Kuvassa näkyy kolme projektia sekä niihin liittyviä metriikoita. Kuva on peräisin Hackystatin tutoriaalista. [2]

testejä, sekä niiden tulokset. Tästä datasta voi myös saada tarkempia visualisaatioita. Näin yksityiskohtainen data yksittäisistä kehittäjistä saa jotkut kehittäjät tuntemaan olonsa epämukavaksi, mutta se on välttämätöntä joidenkin havaintojen tekemiseksi. Esimerkiksi ketterien menetelmien periaate on, että ohjelmistoja tulisi koostaa aikaisin ja usein, ja tällainen data voi auttaa selvittämään, kuinka hyvin tätä noudatetaan. Lisäksi Hackystatin tapauksessa on sama ongelma kuin Leapissa: uusien mittausten lisääminen vaatii huomattavasti enemmän työtä kuin manuaalisen kirjanpidon tapauksessa.

Viime vuosina kaupalliset ohjelmistoanalytiikkatyökalut ovat alkaneet yleistyä. Tyypillisesti ne koostuvat kolmesta perusosasta: konfiguraationhallintajärjestelmästä, käännös/koostamisjärjestelmästä ja virheenkirjausjärjestelmästä. Näissä työkaluissa on kaksi huomattavaa etua. Ensinnäkin datan keruu on täysin automatisoitu ja data on valmiiksi saatavilla, työkalut vain käyttävät analysointitekniikoita dataan ja näyttävät sen valmiissa käyttöliittymässä. Koska data on kerätty automaattisesti tietovarastosta, ylimääräinen vaiva kehittäjille ja hallinnolle on pieni. Lisäksi data ei yleensä ole kyseenalaista, sillä se keskittyy tuotteen ominaisuuksiin, eikä kehittäjien työtapoihin. Tällaisen järjestelmän käyttöönotto on melko helppoa ja vaivatonta, mutta se ei pysty vastaamaan moniin kysymyksiin, kuten esimerkiksi kehittäjien työtapoihin liittyviin kysymyksiin. Ohjelmistoanalytiikassa joudutaankin tekemään monia kompromisseja, esimerkiksi automatisaation, kehittäjille tai hallinnolle aiheutuvan vaivan, käyttöönottamiskynnyksen (joka voi olla sosiaalinen) ja tekniikan tai

teknologian yleisyyden, eli millaista analytiikkaa se tukee, välillä. Ohjelmistoanalytiikka ei kuitenkaan ole yhtenäistymässä yhtä ainoaa lähestymistapaa kohti eivätkä uudemmat lähestymistavat välttämättä ole parempia kuin vanhat, esimerkiksi PSP, vaan ne tekevät erilaisia kompromisseja. Hybridi lähestymistapa, jossa yhdistetään automaattisen datan keräyksen ja analysoinnin parhaat puolet sekä tarkasti valittu manuaalisen datan keräys, voisi lisätä analyysin vaikutusta kasvattamatta kehittäjille aiheutuvaa vaivaa liian suureksi.

2.2 Esimerkkejä ohjelmistoanalytiikan sovelluksista

Ohjelmistoanalytiikalla tulisi olla teoreettisen tutkimuksen lisäksi myös käytännön sovelluksia, joilla saadaan konkreettisia hyötyjä, kuten pienentynyt työmäärä tai ohjelmiston parantunut laatu. Esimerkiksi Microsoftin projektit StackMine [37] ja CODEMINE [17] ovat ohjelmistoanalytiikan käytännön sovelluksia, joilla on saatu aikaan konkreettisia hyötyjä. StackMine on tarkoitettu analysoimaan automaattisesti käyttäjiltä kerättyjä suoritustallenteita, joita on niin paljon, että niiden kaikkien analysoiminen käsin olisi käytännössä mahdotonta. Se siis keskittyy prosessien sijaan ohjelmien, ja niiden tehokkuuden analysointiin. Projektilla saatiin lopulta vähennettyä huomattavasti analysointiin kuluvaa työmäärää, mutta sen aikana huomattiin, että on tärkeää keskittyä tuottavuuden kannalta olennaisiin asioihin, eikä välttämättä akateemisesti kiinnostaviin yksityiskohtiin. CODEMINE taas on Microsoftin käyttämä datan keräämis- ja analysointialusta, joka kehitettiin, kun huomattiin, että eri projektien tarpeisiin tehdyillä työkaluilla oli yhteneväisyyksiä.

2.2.1 StackMine -projekti

Artikkelissa Software Analytics in Practice [37] käsitellään sitä, millaista ohjelmistoanalytiikan tulisi olla ollakseen hyödyllistä ja esimerkkinä annetaan StackMine -projekti, jossa oli tarkoitus etsiä datanlouhinnan avulla pullonkauloja ohjelmien tehokkuudesta. Jos ei mainita toisin, tässä aliluvussa viitataan kyseiseen artikkeliin. Vaikka ohjelmistoanalytiikasta on ollut paljon tutkimusta, vain pieni osa siitä on käsitellyt sen vaikutusta käytäntöihin ohjelmistokehityksessä. Tavoitteena olisi, että analytiikalla saadaan hyödyllistä tietoa tietynlaisissa tehtävissä työskenteville kehittäjille, tai että sen avulla voisi suunnitella konkreettisia keinoja saada tehtävä tehtyä paremmin. Esimerkiksi kehityksen tuottavuutta, ohjelmiston laatua tai käyttäjäkokemusta tulisi voida parantaa analytiikan tuottamien neuvojen avulla. Vaikka ohjelmistoanalytiikassa käytetään todellista maailmaa kuvaavaa dataa, joka usein myös elää ja kehittyy ajan mukana (esim. avoimen lähdekoodin projektien tietovarastot), on sen hyödyntämisessä huomattavia haasteita. Esimerkiksi, mistä tietää, että analyysi tuottaa hyödyllistä tietoa tai käytännöllisiä neuvoja, tai että

edes käyttää oikeaa dataa vastatakseen kysymyksiin, joista kehittäjät välittävät?

Tehokkuusongelmien jäljitys on noussut esiin vasta viime aikoina, mikä johtuu pääosin infrastruktuurista, joka mahdollistaa suoritustallenteiden (engl. execution trace) keräämisen valtavalta käyttäjämäärältä [21]. Yksi sellainen järjestelmä on Microsoftin käyttämä PerfTrack, joka mittaa järjestelmän vasteaikaa käyttäjän toimiin. Esimerkiksi kun käyttäjä painaa Explorerissa valikon valintaa luodakseen uuden kansion, PerfTrack mittaa ajan, joka kestää siihen kunnes käyttäjä saa jonkinlaista palautetta. Mikäli tähän kuluu tietyn rajan yli menevä aika, järjestelmä lähettää suoritustallenteen takaisin Microsoftille analysoitavaksi. Näitä tallenteita on valtava määrä, paljon enemmän kuin mitä tehokkuusanalyytikot voivat tutkia, minkä takia StackMineä alettiin kehittää. Ollessaan yhteydessä Microsoftin kanssa artikkelin kirjoittajat totesivat suureen määrään oikeista sovelluksista kerättyihin tallenteisiin perustuvan analyysin olevan nouseva tutkimusaihe. Koska kirjoittajilla oli taustaa koneoppimisesta, he lähestyivät ongelmaa tästä näkökulmasta. Koneoppimisen näkökulmasta he saivatkin hyviä tuloksia, mutta ne eivät juuri kiinnostaneet Microsoftin tiimiä. Tämän takia tutkijat alkoivat työskennellä läheisemmin tehokkuusanalyytikkojen kanssa tunnistaakseen oleelliset ongelmat. Lopulta he käyttivät kehittämäänsä StackMineä tunnistamaan vaikutukseltaan suuria ohjelman ajopatterneja suuresta määrästä tallenteita. Tunnistaminen aloitettiin kymmenellä oleellisimmalla tapahtumatyypillä, ja analyytikkojen palautteen perusteella algoritmiin lisättiin uusia tapahtumatyyppejä. Datan kohinan vähentäminen on tärkeää, koska kohinainen data voi huonontaa algoritmin tehokkuutta. StackMinessä käytettiin pullonkaulojen kattavuutta mittaamaan patternien tehokkuutta, joka määriteltiin patterneihin liittyvien aikajaksojen suhteena kaikkien kiinnostuksen kohteitten (engl. ROI, region of interest) yhteenlaskettuun aikaan. Aluksi tässä saatiin todella huonoja tuloksia ja syyksi paljastui eräät erittäin pitkät tallenteet, jotka johtuivat tietokoneen menemisestä pitkään valmiustilaan. Näiden suodattaminen pois paransi kattavuutta huomattavasti.

Ohjelmistoanalytiikassa käytettyjen tekniikoiden kehittäminen on yleensä iteratiivinen prosessi, jossa ajoissa saatu palaute on erittäin tärkeää. Yhtenä säännöllisen palautteen hyötynä saatiin määriteltyä malli tallenteiden patternien jakamiseen samankaltaisiin ryhmiin, joita käytettiin esittämään tehokkuusongelmia. Yhteistyössä analyytikkojen kanssa mallia saatiin kuitenkin parannettua niin, että ryhmiin jaottelu toimi riittävän hyvin, sekä osattiin valita arviointikriteerit oikein.

StackMine -projektin tuloksena käyttöjärjestelmän järjestelmäkutsuista syntyneiden pinotallenteiden (engl. stack trace) analysointiin tarvittu työmäärä on suorituskykyä analysoivan tiimin mukaan pienentynyt 90 prosenttia. Projektissa opittiin, että data-analytiikan näkökulmasta hyvätkään tulokset eivät välttämättä auta kehittäjiä käytännön työssä. On tärkeää tunnistaa oleellisimmat ongelmat, joiden

ratkaiseminen parantaisi esimerkiksi tuottavuutta tai laatua, ja käyttää oikeita datajoukkoja niiden ratkaisemiseen. Kuten projektissa havaittiin, läheinen yhteistyö tutkijoiden ja kehittäjien välillä, sekä nopeasti saatava palaute auttaa tässä huomattavasti. Käytännössä datan ymmärtämisen voi jakaa kolmeen vaiheeseen: datan valitseminen, suodattaminen ja tulkitseminen. Datan tulkitsemisessa tarvitaan aihealueeseen liittyvää tietoa ja osaamista, kuten termit, käsitteet ja yleiset perusteet. Näistä joutuu yleensä ottamaan selvää kehittäjiltä, ja heistäkin vain harvat tietävät, mikä on tärkeää mihinkin ongelmaan liittyen. Lisäksi on tärkeää valita data, joka liittyy ongelmaan, epäoleellisen datan analysointi voi vaikeuttaa analysointia tai kuluttaa suuria määriä laskentatehoa. Datan suodattaminen on myös oleellista, sillä siinä voi olla vikoja, jotka voivat johtaa vääriin lopputuloksiin. On myös huomattavaa, että todellisten ongelmien löytäminen vaatii usein skaalautuvia analytiikkatyökaluja ja niiden mukauttamista, mikä vaatii tietämystä aihealueesta. Mukauttamiseen kuuluu kohinaisen ja epäoleellisen datan suodattaminen, datan sisäisten relaatioitten täsmentämistä ja kokeellista ja heuristista ohjausta jolla algoritmit saadaan toimimaan paremmin virheellisenkin datan tapauksessa. StackMinessä skaalautuvuus ja muokattavuus olivat erittäin tärkeitä, ilman hajautettua laskentaa se ei olisi pystynyt käsittelemään tarvittavaa datamäärää, ja ilman mukauttamista analyytikkojen tietämystä ei olisi voinut käyttää hyödyksi.

2.2.2 CODEMINE -projekti

Ellei mainita toisin, tässä aliluvussa käsitellään Jacek Czerwonkan et. al. artikkelia CODEMINE: Building a Software Development Data Analytics Platform at Microsoft [17] ja siinä kuvattua järjestelmää.

Aikaisin saatu, luotettava ja riittävän usein päivittyvä data antaa insinööreille ja hallinnolle mahdollisuuden tehdä dataan pohjautuvia päätöksiä. Tämä puolestaan parantaa kehitysprosessia, mikä näkyy parantuneena ohjelmiston laatuna ja aikataulussa pysymisenä. Microsoftilla useat tiimit käyttävät dataa esimerkiksi prosessien parantamiseen, trendien ja kehityksen terveyden seuraamiseen ja riskien arviointiin. Arvioidessaan olemassaolevia työkaluja Microsoftin tiimit huomasivat, että vaikka jokainen ratkaisu on ainutlaatuinen tarkoitetussa tehtävässään, niissä on yhteisiä sisääntuloja ja ulostuloja sekä metodeja. Esimerkiksi suurin osa työkaluista käyttää samanlaista dataa. CODEMINE on Microsoftin sisäiseen käyttöön kehittämä datan analysointialusta, joka kerää ja analysoi ohjelmiston kehitysprosessin tuottamaa dataa. Sen kehittäminen alkoi loppuvuodesta 2009, tarkoituksena oli toteuttaa yhteinen alusta analytiikkaa varten. Sen käyttö yleistyi nopeasti ja nykyisin sitä käytetään kaikissa isoissa Microsoftin tuoteryhmissä. Projekti ei ollut vain akateeminen tutkimus, vaan järjestelmää käytetään Microsoftilla ja sillä on satoja käyttäjiä. Lähdekoodeja sisältävät tietovarastot ovat muutoksien määrässä ja koossa mitattu-

na suurin datan lähde. Niistä saadaan tietoa lähdekoodiin liittyvistä asioista, kuten koodin tilasta kertovaa dataa, sekä dataa meneillään olevista muutoksista. Ensin mainittuihin kuuluu esimerkiksi kooditiedostojen koko, kommenttien ja itse koodin määrä ja vastaavia mittareita, sekä jälkimmäiseen muutoksia, haara johon koodi kuuluu, ja tiimin tuotanto ajan mittaan.

Erilaiset projektinhallintatyökalut, kuten virheen raportointijärjestelmät tai Jira, ovat toinen tärkeä datan lähde. Niissä on yleensä tallennettuna kehitettävän ohjelman ominaisuuksia ja virheitä, jotka usein liittyvät läheisesti ohjelmakoodin muutoksiin. Data koosteista (engl. builds) taas kuvaa lopullista ohjelmistotuotetta, ja koodin katselmukset ja testit täydentävät kuvaa suunnittelun etenemisestä. Lisäksi tiedot organisaatiosta ja prosesseista (esim. julkaisuaikataulut ja virstanpylväät (engl. milestone)) ovat osa CODEMINEä. CODEMINE:llä on standardoitu rajapinta, jonka kautta ohjelmat pääsevät käsiksi dataan. Käyttäessään dataa ohjelmat kysyvät ensin, onko haluttu data olemassa, ja missä se on, ja sitten kyselevät dataa datamallin kautta. Tämä on ensisijainen tapa päästä dataan käsiksi, mutta tarvittaessa myös yksittäisten datalähteiden rajapintoihin pääsee käsiksi. Tällaisessa järjestelmässä tietoturva on tärkeää, ja erilaisiin välimuisteihin tallennettuun dataan pääsyä tulee voida valvoa samalla tavalla kuin raakadataan pääsyä.

Järjestelmää toteutettaessa ilmeni kolme pääasiallista datan käyttötapaa: datanlähteenä tuotantotiimin prosessiin liittyvälle raportointityökalulle, yksittäiseen kysymykseen vastaavalle kustomoidulle analyysille tai uuden tutkimuksen mahdollistaminen. Raportointityökalut vaativat datan tuoreutta ja sen saamisen ja analyysin luotettavuutta, sekä tehokasta datan hakua. Esimerkkinä uudesta tutkimuksesta Christian Bird et al. tutkivat versionhallinnassa olevien haarojen muodostaman rakenteen vaikutusta kehitykseen [12]. Kustomoidun analyysin tapauksessa taas tärkeintä on, että data on ylipäätään saatavilla, ja että siihen on helppo päästä käsiksi. Esimerkiksi CRANE [16] on virheenennustus-, muutosriskianalyysi- ja testauksen priorisointijärjestelmä, joka on rakennettu CODEMINE-infrastruktuurin päälle. CODEMINE seuraa jatkuvasti muutoksia tietovarastoissa, ja yhdistää ominaisuudet ja virheet yms. toisiinsa. CRANE käyttää tätä dataa ja automaattisesti tiedottaa tiimeille esim. virheistä ja mahdollisista riskeistä, sekä suosittelee keinoja niiden minimoimiseksi. CRANE tulkitsee dataa esimerkiksi kattavuudesta, riippuvuuksista ja aiemmista virheistä ja yrittää suositella tiettyihin toimintoihin kohdistuvia testejä, joilla on parempi mahdollisuus löytää virheitä.

On kuitenkin hyvä kysymys, onko koodin kattavuus tehokas mittari, ja onko "riittävää" testikattavuutta, jonka jälkeen ei tarvitse testata. Asiaa analysoidessa selvisi, että haara- ja lausekattavuudella oli vahva positiivinen korrelaatio julkaisun jälkeisen virhemäärän kanssa. Tälle odotusten vastaiselle relaatiolle on useita syitä: Koodin kattavuus ei takaa, että koodi toimisi oikein, ja 100% kattavuus ei tarkoita,

ettei järjestelmässä olisi virheitä, sillä ne voivat olla muissa kuin testatuissa skenaarioissa. Lisäksi kun korjataan virhe, yleensä sitä varten kirjoitetaan testitapaus. Siksi koodissa, johon ollaan tehty paljon muutoksia, voi olla hyvä kattavuus. Tämän takia koodin kattavuus yhdistettynä kompleksisuuteen on parempi mittari koodin laadulle.

CODEMINE:n onnistumisen taustalla yksi tärkeä ominaisuus oli itsenäisen ilmentymän (engl. instance) tekeminen jokaiselle tuotantotiimille, missä auttoi kyky osittaa ja mahdollisuus rajoittaa dataan pääsyä, sekä siirtää osia infrastruktuurista. Toinen oli yhdenmukaiset rajapinnat datan analysointia varten. Alaspäin yhteensopivuus ja tietomallin samana pysyminen helpottavat työkalujen uudelleenkäyttöä. Prosessi-informaation, kuten julkaisuaikataulun, tiimien rakenteen ja muun vastaavan informaation varastoiminen on oleellista, sillä sen avulla voidaan saada eri tilanteiden asiayhteys paremmin selville, esimerkiksi se voi auttaa selvittämään virheiden määrässä olevaa piikkiä. Lisäksi järjestelmää tulee voida laajentaa, ja työkalujen pitäisi tulla toimeen vähemmällä datalla, jos sitä ei ole saatavissa jostain järjestelmän instanssista. Tietoturva on myös erittäin oleellista, dataan tulisi olla pääsy vain niillä, jotka pääsivät käsiksi myös alkuperäiseen dataan.

CODEMINE-projektissa huomattiin myös, että kun dataa on helposti saatavilla, uusia käyttötapoja ilmenee, kuten koostamisen optimointi. Lisäksi uuden tutkimuksen mahdollistaminen poikii uusia ratkaisuja, jotka puolestaan ratkaisevat ongelmia, joka taas mahdollistaa lisää tutkimusta.

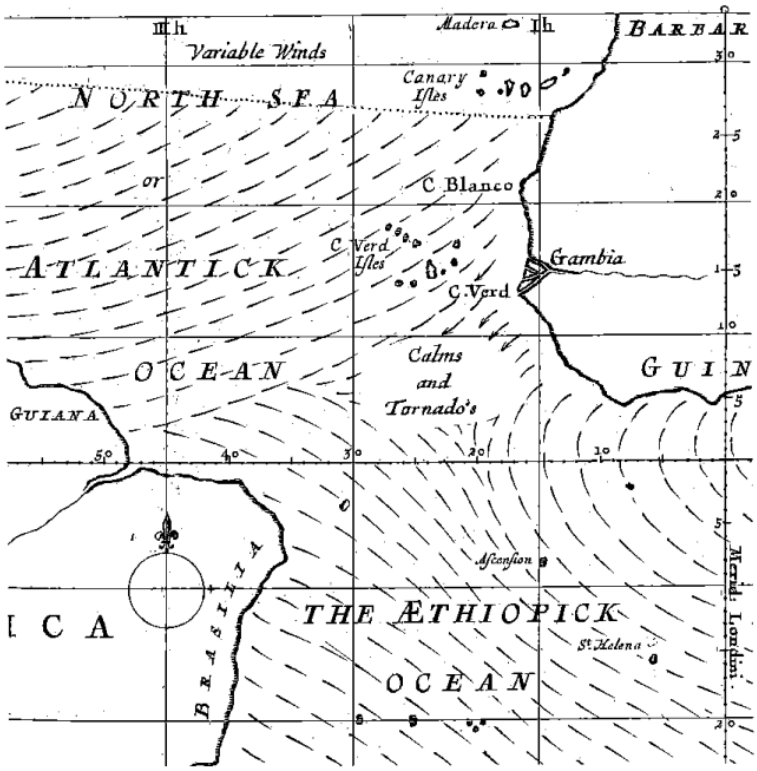
3. DATAN VISUALISOINTI

Tässä luvussa käsitellään Tuften kirjaa The Visual Display of Quantitative Information [35]. Kirja käsittelee datan visualisointia, ja sitä, millainen laadukkaan kuvaajan tulisi olla. Kuvaajan tulisi näyttää data, saattaa katsoja ajattelemaan sisältöä metodologian, graafisen suunnittelun tai tekniikan sijaan, välttää datan sanoman vääristämistä, esittää paljon numeroita pienessä tilassa, saada suurista datajoukoista koherentteja, rohkaista vertaamaan keskenään kaaviossa esitettyä dataa ja paljastaa data eri detaljitasoilla. Sillä tulisi myös olla kohtuullisen selvä tarkoitus ja olla läheisesti integroituna tilastollisiin ja sanallisiin kuvauksiin datajoukosta. Grafiikka "paljastaa" dataa, se voi olla havainnollisempi kuin perinteiset tilastolliset laskutoimitukset. Tästä huolimatta kuvaajat ovat vain niin hyviä kuin data, jota niiden luomiseen käytetään.

Kirjassa käsitellään myös datan visualisoinnin historiaa. Datakartat (engl. data map) ovat melko uusi asia, ne keksittiin vasta 1600-luvulla, vaikka yksityiskohtaisia karttoja oli piirretty vuosisatoja aiemmin. Vaikka karttojen piirtäminen oli lähellä tilastollisia kuvaajia, kukaan ei ollut vielä tehnyt abstraktiota, jossa kartalle asetetaan joku mitattu suure esimerkiksi kaupungin sijaan, puhumattakaan pituus- ja leveysasteiden korvaamisesta muilla suureilla kuten ajalla ja rahalla. Yksi ensimmäisistä datakartoista on kuvassa 3.1 oleva 1600-luvun lopulta oleva kartta, jossa kuvataan tuulia ja monsuuneja maailman merillä.

Aikasarjakuvaaja (engl. time series plot) on yksi yleisimmistä käytetyistä kuvaajista, ja se alkoi yleistyä tieteellisissä teksteissä vasta 1700-luvun lopulla. J.H. Lambert ja William Playfair olivat nykyisen graafisten kuvaajien kaksi suurta keksijää. Ensimmäiset tunnetut talouteen liittyvää dataa kuvaavat aikasarjakuvaajat, kuten esimerkiksi kuvassa 3.2 oleva kuvaaja, ilmestyivät vasta Playfairin kirjassa The Commercial and Political Atlas. Aikasarjakuvaajien ongelmana on kuitenkin se, että ajan kuluminen ei välttämättä ole hyvä selittävä muuttuja. Seuraava askel oli abstraktimmat kuvaajat kuten kuvaajat, joissa muuttuja y esitettiin muuttujan x funktiona.

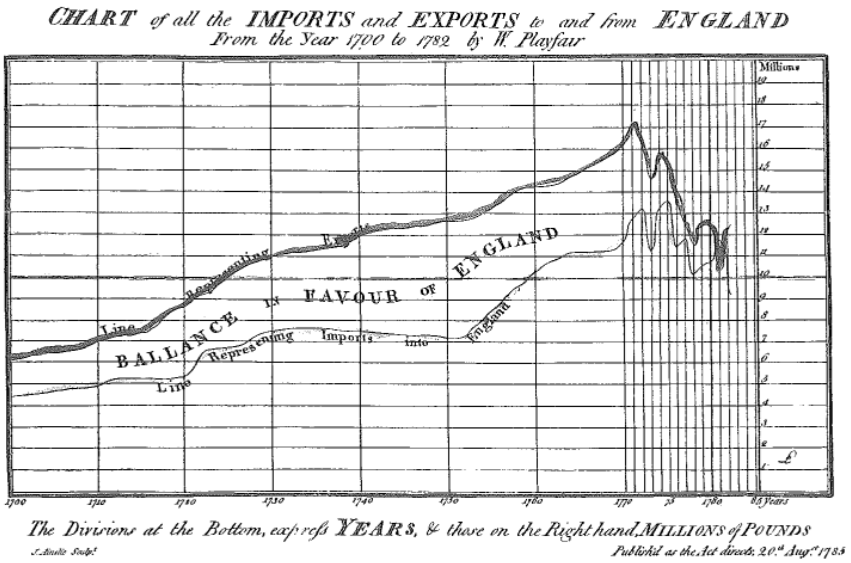
Graafisten kuvaajien tulisi kuitenkin olla totuudenmukaisia, eikä niiden tulisi vääristää dataa. Esimerkiksi pylväsdiagrammin pylväiden aloittaminen muualta kuin nollasta usein vääristää kuvaajaa. Kuvaajassa esitettyjen numeroita vastaavien suureiden, kuten pinta-alan, tulisi olla suoraan verrannollisia datassa oleviin numeerisiin



Kuva 3.1: Edmond Halleyn piirtämä datakartta tuulista ja monsuuneista. Kuva on peräisin Tuften kirjan sivulta 23. [35]

arvoihin. Lisäksi kuvaajan pitäisi olla yhtenevä, esimerkiksi asteikon tulisi olla samanlainen joka osassa kuvaajaa. Erilaisen aikavälin käyttäminen eri osissa kuvaajaa vääristää dataa. Usein nähdään kuvaajia, joissa koko muuttuu suureeseen verrannollisesti sekä pysty- että vaakasuunnassa, vaikka itse suure on yksiulotteinen. Tällöin suureen verrannollisuus pinta-alaan ei säily.

Kirjassa sanotaan kuvaajan suunnittelun pääperiaatteen olevan se, että ennen kaikkea sen tulee näyttää data. Suurin osa kuvaajassa olevan "musteen" määrästä tulisi kulua datan ilmaisemiseen. Tiettyyn rajaan asti on sitä parempi mitä suurempi osa kuvaajassa olevasta musteesta kuluu datan ilmaisemiseen, ja siksi "pyyhkimisperiaatteena" on, että järkevyyden rajoissa tulisi pyyhkiä dataa kuvaamaton muste. Esimerkki tämän kuvaajaa selkeyttävästä vaikutuksesta nähdään kuvassa 3.3. Lisäksi redundanssia datan ilmaisemiseen kuluvassa musteessa tulisi pääosin välttää. Esimerkiksi varjostettu ja nimetty pylväs pylväsdiagrammissa kertoo korkeuden monella eri tavalla (oikean ja vasemman viivan korkeus, varjostuksen korkeus, ylemmän horisontaalisen viivan korkeus ja nimen arvo ja paikka), mikä on tarpeetonta. Joissain tapauksissa redundanssista on kuitenkin hyötyä: se voi antaa monimutkaiselle kuvalle asiayhteyden ja järjestyksen sekä helpottaa vertailujen tekemistä. Esimerkiksi maailmankartan sisältävässä kuvaajassa voi olla parempi näyttää osa datasta



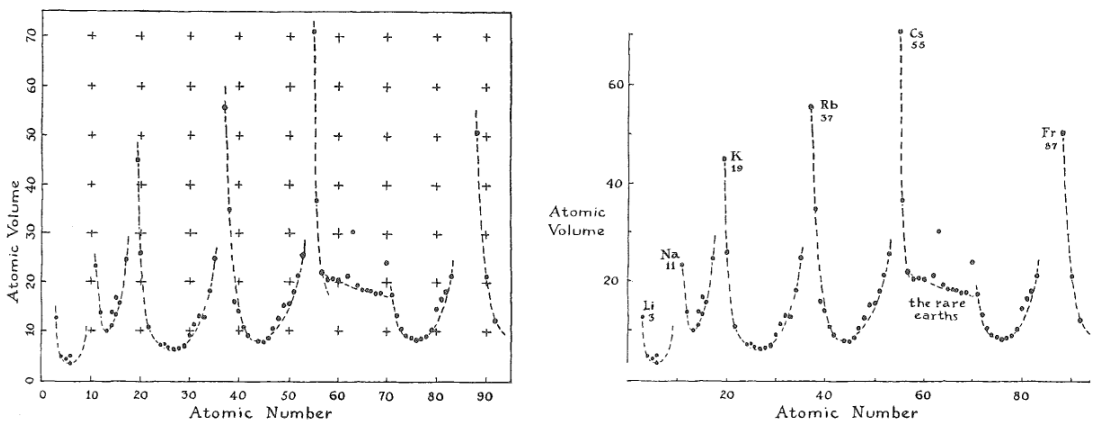
Kuva 3.2: Playfairin Englannin kauppatasetta kuvaava aikasarjakuvaaja. Kuva on Tuften kirjan sivulta 32. [35]

useaan kertaan, ettei silmä joudu hyppimään kuvaajan puolelta toiselle vertaillessa kuvaajan reunan eri puolilla olevia paikkoja. Lisäksi on tärkeää muokata kuvaajaa useaan kertaan ja kokeilla, miltä eri versiot näyttävät.

Tuften kirjassa varoitetaan "kaaviojätteestä" (engl. chart junk) kuten turhasta koristelusta tai liian näkyvästä ruudukosta. Moire-interferenssikuvio eli kuvio, joka saadaan aikaan päällekkäin asetetuilla ruutu- tai raitakuvioilla, on asia, mitä kaavioissa tulisi välttää. Se kiinnittää helposti huomion, ja sen liiallinen käyttäminen voi tuntua häiritsevältä luettaessa muuta osaa sivusta, ja se tekee kaavion hahmottamisesta vaikeaa. Tummat ruudukot ovat myös kaaviojätettä, ne eivät välitä informaatiota ja sotkevat kuvaajaa. Esimerkiksi harmaa ruudukko on parempi kuin musta. Lisäksi koristelua ei saisi olla liikaa, eikä sen tulisi ottaa kaaviota haltuunsa.

Silmillä pystyy erottamaan huomattavan määrän yksityiskohtia pieneltä alueelta. Yhden neliösentin alueelta pystyy erottamaan noin 100 pistettä. Kartoissa esitetään rutiininomaisesti varsin pieniä yksityiskohtia, jopa 0,1 mm suuruisia. Vaikka kuviin voidaan siis mahduttaa suuria määriä tietoa, useimmissa kuvaajissa datan tiheys on silti huomattavan pieni. Kuvaajissa tulisikin järkevyyden rajoissa käyttää mahdollisimman suurta datan tiheyttä ja datamäärää. Datan tiheyttä voidaan datamäärän kasvattamisen sijaan lisätä myös pienentämällä kuvaajan kokoa. Yksi tapa hyödyntää tätä on käyttää sarjaa pieniä kuvaajia, joissa on samat muuttujat indeksoituna jonkun toisen muuttujan muutoksilla. Esimerkiksi kyseessä voi olla tunneittainen kartta ilmansaasteista.

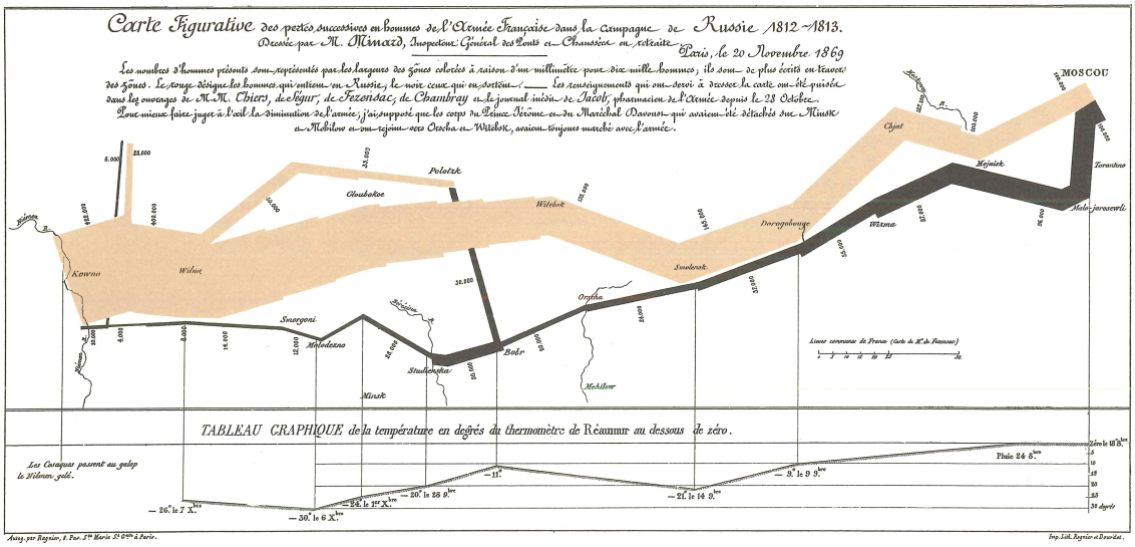
On vaikeaa määritellä periaatteita, joiden avulla luoda loistavia kuvaajia, mutta joitain yleisiä ohjenuoria voidaan ehdottaa. Esimerkiksi Minardin piirtämää kuvaa-



Kuva 3.3: Tuften kirjassa käytetty esimerkki "pyyhkimisperiaatteesta". Kuvasta on pyyhitty taustalla olleet rastit, sekä osa akseleista ja akseleilla olevista numeroista, ja lisätty selventäviä tekstejä. Kuvat löytyvät kirjan sivuilta 102 ja 105. [35]

jaa Napoleonin armeijan marssista Moskovaan (kuva 3.4) pidetään hyvänä visualisaationa. On tärkeää, että kuvaajan malli ja formaatti ovat oikeanlaisia, sekä tekstiä ja piirroksia on hyvä yhdistää, pelkän kuvaajaan tekstistä viitaamisen sijaan. Mittasuhteitten ja mittakaavojen tulisi olla selviä, ja yksityiskohdat tulisi esittää sopivalla tarkkuudella. Kokonaisuuden tulisi olla ymmärrettävissä, mutta kaavion katsojia ei tulisi aliarvioida. Lopuksi olisi tärkeää välttää sisällötöntä koristelua.

Datan esittämisessä täytyy myös valita tekstin, taulukon ja kuvaajan väliltä, ja usein kahta näistä kannattaa yhdistää. Pelkkä teksti on huono tapa esittää useampaa kuin kahta numeroa, koska se vaikeuttaa datan sisällä tapahtuvia vertailuja. Taulukot ovat paras tapa näyttää tarkkoja lukuarvoja, ja niitä kannattaa käyttää pienien datasettien tapauksessa kuvaajien sijaan. Taulukko on lähes aina parempi keino näyttää dataa kuin piirakkadiagrammi. Taulukot toimivat hyvin myös silloin kun tarvitsee vertailla monia toistensa vierellä olevia arvoja. Kuvaajiin on lähes aina hyödyllistä kirjoittaa dataa selittävää tekstiä, sekä nimetä kiinnostavia kohtia kuvaajasta. Lisäksi kuvaajasta olisi hyvä puhua tekstissä lähellä kuvaajaa sen sijaan, että kuvaan viitattaisiin eri osassa tekstissä, jolloin lukija joutuu vilkuilemaan edestakaisin. Koska kuvaajissa olevat tekstit ovat yleensä melko lyhyitä, on mahdollista käyttää pientä fonttia ilman, että se saa lukijan väsymään hankalalukuisuuteen. Monet pienet yksityiskohdat kuten epäselvien lyhenteiden välttäminen, selventävät tekstit, värisokeaystävällisesti valitut värit, selkeät fontit ja sekä isojen että pienten kirjaimien käyttö tekevät kaaviosta selkeämmän. Kuvaajassa olevat viivat kannattaa yleensä pitää ohuena, ja mittasuhteissa kannattaa suosia leveämpää kuin kapeampaa kuvaajaa, ellei datan takia ole luonnollista käyttää jonkin muun muotoista kuvaajaa.



Kuva 3.4: Minardin kuvaaja Napoleonin armeijan marssista Moskovaan. Kuva esitetään Tuften kirjassa esimerkkinä hyvin tehdystä kuvaajasta sivulla 176. [35]

4. OHJELMISTODATA, SEN KERÄYS JA KÄYTETYT TYÖKALUT

Digilen rahoittaman Need for Speed -tutkimuskonsortion tarkoituksena on edistää suomalaisten ohjelmistoalan yritysten kilpailukykyä [18]. Esimerkiksi tässä työssä tehdyillä visualisaatioilla pyritään nopeuttamaan tuotekehitysprosesseja, jotta uusia tuotteita tai ominaisuuksia saataisiin tehtyä nopeammin laadun kärsimättä. Ohjelmistokehityksessä on alettu siirtyä ketteriin menetelmiin, ja ne ovat olleet parannus perinteisiin ohjelmistokehityksen malleihin (esimerkiksi vesiputousmalliin) verrattuna, varsinkin pienien tiimien tapauksessa [19]. Kehityksen ja käyttöönoton nopeutuminen vaatii uusien ohjelmistoversioiden testaamiseen ja vikojen korjaamiseen kuluvan ajan lyhentämistä, minkä takia pyritään jatkuvaan integraatioon, nopeaan tuottamiseen ja käyttöönottoon. Esimerkiksi testaukseen kuluvaa aikaa voidaan vähentää testien automatisoinnilla tai jaetulla ja inkrementaalisella testauksella, jolloin testataan ainoastaan edellisestä versiosta muuttuneita osia. Jatkuvassa integroinnissa taas pyritään saamaan ohjelmasta mahdollisimman usein, esimerkiksi päivittäin, uusia versioita, jolloin asiakkailta saadaan nopeammin palautetta, ja virheiden korjaukset pääsevät nopeammin asiakkaiden käyttöön.

Nopeatahtisessa kehityksessä on hyödyllistä, että projektin tai ohjelmiston tämänhetkisestä tilasta saadaan nopeasti tietoa. Esimerkiksi automaattisesti päivittyvät visualisaatiot projektin tilasta voivat olla intuitiivinen ja nopea tapa saada kuva asiasta. Yhteistyöyhtiöltä saadusta versionhallinta- ja projektinhallintadatasta visualisoitiin esimerkiksi versionhallintaan vietyjen muutoksien määrää, koodirivien määrää, projektinhallintatyökalussa kullakin ajanhetkellä olevien issueitten määrää, sekä esimerkiksi virheiden määrää ja elinkaarta. Koska valmiit visualisointityökalut eivät soveltuneet tarkoitukseen, päädyttiin tekemään omia visualisaatioita matalan tason kirjastojen avulla. Näiden visualisointien tekeminen yksitellen jokaista datajoukkoa varten on kuitenkin työlästä, minkä takia päätettiin kehittää oma visualisointikirjasto projektin tarpeisiin. Tässä luvussa käydään läpi datan lähteitä ja sen keräämistä, sekä käydään läpi käytetyt työkalut.

4.1 Ohjelmistokehityksen datalähteet

Ohjelmistokehityksessä käytetään lähes aina versionhallintajärjestelmiä, joiden tarkoituksena on varastoida ohjelmakoodi ja sen historia. Lisäksi useimmissa pro-

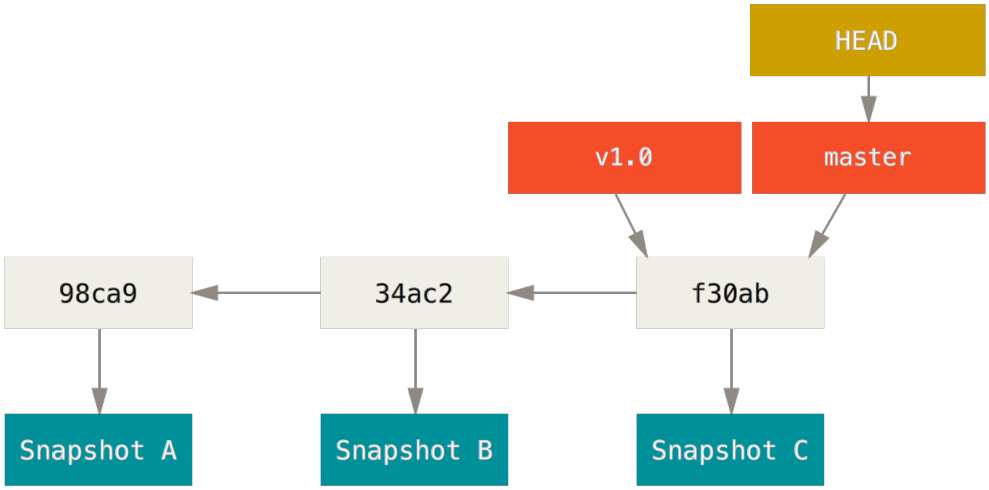
jekteissa käytetään tehtävänhallintajärjestelmää, joiden tarkoitus on toimia apuna projektin organisoimisessa. Esimerkiksi kehittäjille annetut tehtävät eivät unohdu yhtä helposti jos ne on merkitty tehtävänhallintajärjestelmään. Koska version- ja tehtävänhallintajärjestelmät ovat läheisessä yhteydessä kehitysprosessiin, ne ovat usein hyviä datalähteitä kun halutaan ymmärtää kehitysprosessia.

4.1.1 Versionhallintajärjestelmät

Käytännössä kaikissa suuremmissa ohjelmistoprojekteissa käytetään jotain versionhallintatyökalua, joilla pidetään kirjaa muutoksista, ja joilla voidaan palauttaa mikä tahansa koodin (versionhallintaan tallennettu) tila. Lisäksi versionhallintajärjestelmät tallentavat metadataa muutoksista, kuten muutoksen tehneen käyttäjän ja päivämäärän. Versionhallintajärjestelmät voidaan jakaa kahteen luokkaan: keskitetyt ja hajautetut järjestelmät [33]. Keskitetyt versionhallintajärjestelmät ovat perinteinen ratkaisu, jossa kaikki tieto on keskitetyssä tietovarastossa ja kaikki kehittäjät käyttävät suoraan tätä keskitettyä palvelinta. Esimerkiksi CVS ja Subversion ovat tällaisia järjestestelmiä.

Hajautetut versionhallintajärjestelmät ovat keskitettyjä uudempi ratkaisu, jossa jokaisella kehittäjällä on täysi kopio tietovarastosta ja jota käytetään paikallisesti. Myöhemmin paikallisesta tietovarastosta julkaistaan halutut haarat päätietovarastoon, jotta muutkin kehittäjät pääsevät muutoksiin käsiksi. Käytännössä suurin ero keskitettyjen ja hajautettujen versionhallintajärjestelmien välillä on se, että kehittäjät eivät tarvitse jatkuvasti Internet-yhteyttä käyttääkseen versionhallintaa, vaan pelkästään muutosten julkaisemiseen, ja koko projektin historiaa ei pidetä vain yhdessä paikassa. Lisäksi hajautetuissa versionhallintajärjestelmissä muutoksen tekijä voi olla eri henkilö kuin muutoksen vienyt henkilö, minkä vuoksi järjestelmät pitävät näistä kirjaa erikseen.

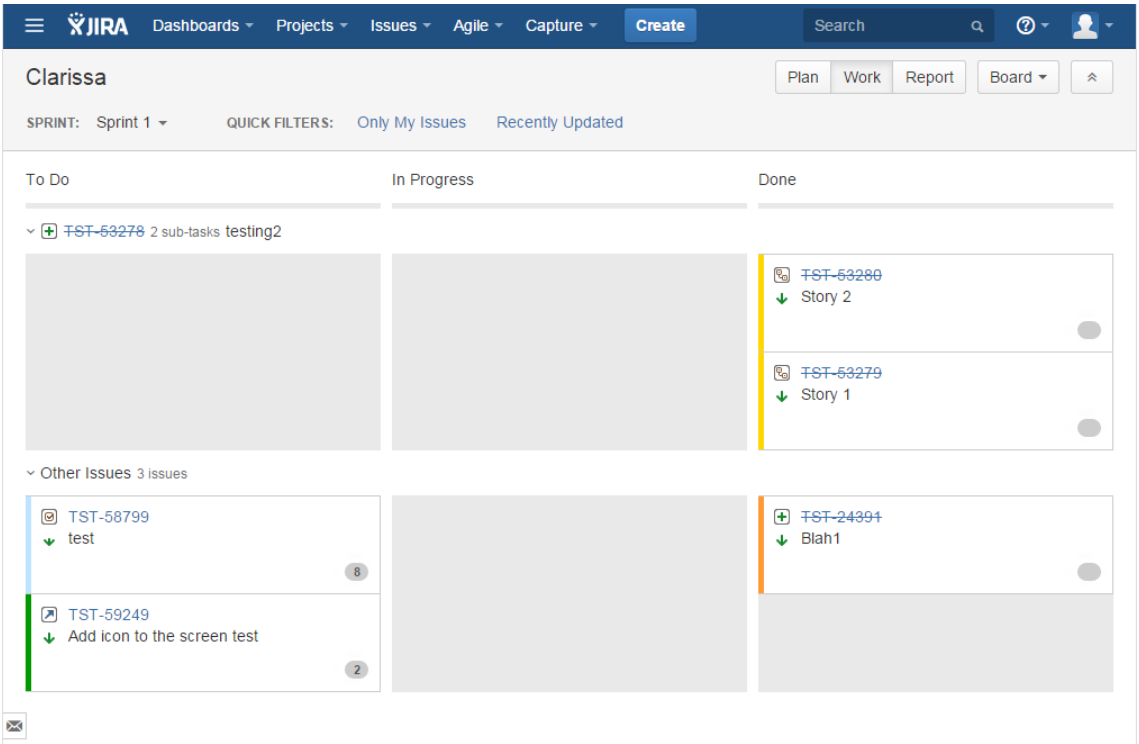
Git on eräs hajautettu versionhallintajärjestelmä, se on avoimen lähdekoodin ohjelma ja se on levinnyt yleiseen käyttöön [15]. Gitin toiminta eroaa huomattavasti esimerkiksi Subversionista. Subversion varastoi tietoa listana tiedostopohjaisista muutoksista. Se siis näkee tiedon joukkona tiedostoja ja niihin ajan mittaan tehtyinä muutoksina. Git tavallaan näkee tiedon eräänlaisena pienenä tiedostojärjestelmänä, ja ottaa vain kuvan (engl. snapshot) sen tilasta. Lisäksi hajautettuna versionhallintajärjestelmänä melkein kaikki Gitin operaatiot ovat paikallisia, esimerkiksi projektin historian selaamiseen ei tarvitse Internet-yhteyttä. Kaikesta Gitissä olevassa tiedosta lasketaan tarkistussummat ja tämän jälkeen siihen viitataan kyseisellä summalla. Tämän takia Gitiin ei voi tehdä muutoksia ilman että järjestelmä havaitsee ne, eikä tiedon korruptoituminen jää huomaamatta. Git myös vain lisää dataa joitain poikkeuksia lukuunottamatta, joten versionhallintaan tallennettua dataa on vaikea menettää.



Kuva 4.1: Kuva Gitin historiasta ja haaroista. Kuva on peräisin kirjan ProGit nettiversiosta. [15]

Gitissä on tietovarasto, 'staging area' ja työhakemisto. Työhakemistossa on erilaisia tiedostoja, joiden ei tarvitse olla lisätty versionhallintaan. 'Staging area' on väliaikaisvarasto, johon lisätään tiedostot, joiden muutokset halutaan tallentaa, ja josta viedään muutokset (engl. commit) itse tietovarastoon. Itse tietovarastosta voidaan työntää halutut muutokset verkkoyhteyden päässä olevaan saman projektin tietovarastoon.

Kuten monissa muissa versionhallintajärjestelmissä, myös Gitissä voi laittaa ns. tageja merkitsemään tiettyjä pisteitä historiassa. Usein niillä merkitään esim. ohjelmasta julkaistavia versioita tai vastaavia tärkeitä kohtia projektin historiassa. Gitin ehkä tärkein ominaisuus ovat kuitenkin haarat ja se, että niiden käyttö on huomattavan helppoa ja kevyttä muihin versionhallintajärjestelmiin verrattuna. Gitissä muutokset ovat käytännössä vain osoittimia Gitin tiedostojärjestelmässä olevaan kuvaan tiedon tilasta ja vanhempiin muutoksiin (jos niitä on). Gitissä haara on vain liikuteltava osoitin johonkin tällaiseen muutokseen, minkä takia uusien haarojen tekeminen on erittäin kevyttä. Toisin kuin monissa muissa versionhallintajärjestelmissä, ei ole tarvetta kopioida tiedostoja ja niiden muutoksia uuteen tietorakenteeseen. Esimerkiksi kun haaraan tehdään uusi muutos, haaraosoitinta vain siirretään osoittamaan uuteen muutokseen. Samoin mihin tahansa muutokseen voidaan helposti tehdä uusi haara vain luomalla siihen osoittava haaraosoitin. Kuvassa 4.1 on esimerkki Gitin historiasta ja haaroista. Yhdistäminen taas toimii joko fast-forward tai tavallisena yhdistämisenä riippuen siitä, onko muutos, johon yhdistettävä haara osoittaa, suora jälkeläinen tämänhetkisen haaran uusimmasta muutoksesta. Tällöin voidaan käyttää fast forward -yhdistämistä jossa haaraosoitinta vain siirretään eteenpäin. Jos



Kuva 4.2: Esimerkki Jirassa olevasta taulusta. Kuva on peräisin Jiran demosta. [8]

molempiin haaroihin on tullut muutoksia, joudutaan käyttämään tavallista yhdistämistä jossa Git yhdistää molemmissa haaroissa yhteisen esi-isän jälkeen tapahtuneet muutokset uuteen muutokseen. Yleensä tämä tapahtuu automaattisesti, mutta jos saman tiedoston samaa osaa on muokattu eri tavalla, seuraa konflikti. Tällöin yhdistäminen täytyy viimeistellä käsin.

Mercurial on toinen hajautettu versionhallintajärjestelmä, joka on monilta osin samanlainen kuin Git. Suurimmat erot ovat historian muutettavuudessa; Gitissä historiaa voi muuttaa toiminnoilla kuten esimerkiksi rebase. Toisin kuin yhdistäminen, se ei luo uutta muutosta vaan lisää muutokset toisessa haarassa oleviin muutoksiin. Tällöin historia ei haaraudu, toisin kuin miten asia todellisuudessa meni kehitettäessä ohjelmaa. Mercurialissa historiaa ei voi muuttaa, ja kaikki muutokset kuuluvat nimettyyn haaraan, eli haaran nimi tallennetaan muutokseen. Tämän takia on helppo selvittää, mihin haaraan muutos alunperin tehtiin, toisin kuin Gitissä. Siitä, kumpi lähestymistapa haaroihin on ohjelmaa kehitettäessä parempi, voi olla montaa mieltä, mutta osoittautui että Gitin lähestymistapa tekee haaroihin liittyvän datan louhimisesta huomattavasti vaikeampaa.

4.1.2 Tehtävänhallintaohjelmistot

Tehtävänhallintaohjelmistoja käytetään yrityksissä laajalti helpottamaan projek-

tien hallinnointia ja toteutusta. Tehtävänhallinnan avulla huolehditaan, että tehtävät tulevat suoritetuksi, eivätkä esimerkiksi unohdu. Lisäksi kirjatessa tehtävät, sekä kuvaukset niiden ratkaisuista, tehtävänhallintaan, tallentuu tieto keskitettyyn ja suhteellisen helposti etsittävään muotoon. Esimerkiksi ohjelmistosta löytyvät virheet tallennetaan usein tehtävänhallintajärjestelmään, ja järjestelmään kirjataan muutokset tehtävän tilassa. Esimerkiksi aloitettaessa virheen korjaaminen, tehtävä asetetaan sitä vastaavaan tilaan, ja kun virhe on korjattu, tehtävä suljetaan. Myös muita tehtäviä, kuten uusia ominaisuuksia tai muita ylläpitotehtäviä, lisätään tehtävänhallintaan vastaavasti.

Koska tehtävänhallintaohjelmistoissa oleva data kuvaa suoraan projektissa tehtyä työtä, se on hyödyllisempää projektin analysoinnin kannalta kuin pelkkä versionhallintadata. Versionhallintadatasta voi olla vaikeaa päätellä suoritettuja tehtäviä, tai yleistä projektin edistymistä, sillä kumpikaan ei ole verrannollinen yksinkertaisiin mittareihin, kuten koodiriveihin, tai vietyjen muutosten määrään. Esimerkiksi tehtävien tai virheiden elinkaaren ja määrän kullakin ajanhetkellä, jotka ovat selvästi kiinnostavampia mittareita kuin koodirivit, saa suoraan tehtävänhallintaohjelmistodatasta. Lisäksi ketteriä menetelmiä käytettäessä pyritään usein tekemään kehitys sprinteissä, jotka ovat yleensä yhdestä kolmeen viikkoon kestäviä tehtäväpaketteja, joiden aikana kyseiset tehtävät pyritään tekemään. Useimmiten tieto sprinteistä on tallennettu tehtävänhallintaan.

Jira [6] on yksi yleisimmin käytetyistä tehtävänhallintaohjelmistoista. Tehtävänhallintaohjelmistona sitä on tarkoitus käyttää ohjelmistokehityksessä esimerkiksi sprinttien, tehtävien ja virheitten kirjaamiseen. Lisäksi siinä on mahdollista käyttää erilaisia tauluja havainnollistamaan tiimin toimintaa, esimerkki tällaisesta on kuvassa 4.2. Sitä käytetään selainpohjaisen käyttöliittymän kautta, ja se tallentaa tiedot palvelimella olevaan SQL-tietokantaan. Tietoihin voi päästä käsiksi joko sivuja läpi käyvän "web crawlerin" avulla, tai suoraan SQL-kyselyillä. Datan kerääminen Jirasta on kuvattu luvussa 4.2.

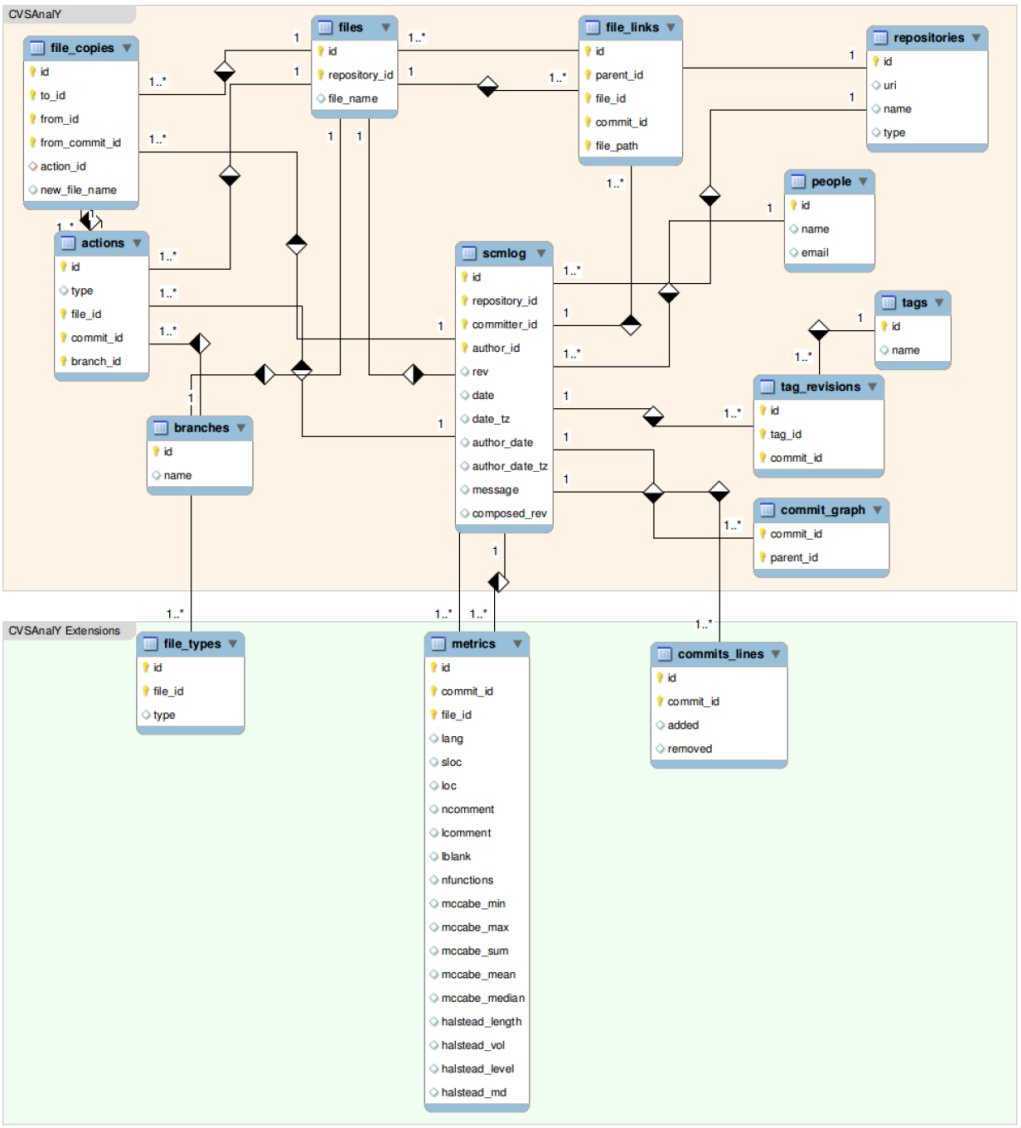
4.2 Datan keräys

Visualisointeihin käytetty versionhallintadata kerättiin CVSAnalY:llä, joka esitellään luvussa 4.2. CVSAnalY ajettiin yhteistyöyhtiön toimesta, ja he anonymisoivat datan CVSAnalY:n tuottamasta tietokannasta, esimerkiksi henkilöiden ja tiedostojen nimet, sekä muu yhtiön kannalta arkaluonteinen data poistettiin. Tämä tietokanta siirrettiin yhtiön tiloista tietokantavedoksena (engl. SQL dump), joka sitten luettiin virtuaalikoneella olevaan MySQL-tietokantaan. Kyseisen tietokannan skeema on kuvassa 4.3. Data ei kuitenkaan ollut sellaisenaan käytettävää, esimerkiksi monilla kehittäjillä oli useita tunnuksia, joilla he veivät muutoksia tietovarastoon, minkä takia heillä oli tietokannassa useita author_id:itä. Siksi yhteistyöyhtiöltä ky-

syttiin, että mitkä ID:t vastaavat kehittäjää, ja Python-komentosarjalla muutettiin ID:t vastaamaan yhtä kehittäjää. Tämän jälkeen dataa haettiin tietokannasta csv-muotoisiin tiedostoihin erilaisten Python-komentosarjojen avulla. Esimerkiksi kokeiltiin selvittää vietyjen muutosten jakautumista tiedostoihin, mutta tässä selvisi, että muutokset ovat hajaantuneet eri tiedostoihin siinä määrin, ettei siitä saanut järkevää dataa. Useimpiin tiedostoihin, joihin oli tehty muutos, muutoksia oli vain yksi kuukauden aikajaksolla. Tämän takia siitä ei saanut järkeviä johtopäätöksiä, vaan korkeintaan sen, että projekti oli hyvin hajoitettu pieniin osakokonaisuuksiin. Muita kokeiluja oli esimerkiksi vietyjen muutoksien jakauma kellonaikojen mukaan, jossa huomattiin piikki kello kahdelta yöllä. Syyksi tähän piikkiin selvisikin eräs Yhdysvalloissa töissä oleva työntekijä, eikä myöhäiset työtunnit.

Versionhallintadatan louhinnassa oli kuitenkin ongelmia, jotka aluksi johtuivat pääasiassa CVSAnalY:n puutteista tai virheistä. Esimerkiksi CVSAnalY ei tukenut vanhempia Gitin versioita, joten yhteistyöyhtiön koneelle jouduttiin päivittämään uudempi versio Gitistä. Lisäksi CVSAnalY:n koodissa oli erilaisia virheitä, yksinkertaisimpana se, ettei luokkaa ollut muistettu tuoda kooditiedostossa, minkä takia ohjelmaa ajettaessa tapahtui poikkeus. Lisäksi osaa CVSAnalY:n lisäosista, kuten Metricsiä ei oltu päivitetty toimivaan uudempien CVSAnalY:n versioiden kanssa. Virheiden korjaamista vaikeutti se, että CVSAnalY:n koodi oli huonosti kommentoitua, eikä sille käytännöllisesti katsoen ollut dokumentaatiota. Lisäksi joitain virheitä ei saatu toistettua ajamalla koodia kaikilla tietovarastoilla, mikä vaikeutti ongelmien korjaamista ja testaamista. Kommunikointi työkalun tekijöiden kanssa auttoi kuitenkin riittävästi, että CVSAnalY saatiin korjattua käyttökelpoiseen kuntoon. Itse versionhallintajärjestelmään tallennetun tiedon rakenne kuitenkin oli ongelmana datanlouhinnassa. Gitissä haara on vain osoitin tiettyyn vietyyn muutokseen, ja osoitin liikkuu sitä mukaan, kun haaraan tulee uusia muutoksia. Mikäli haara yhdistetään toisen haaran kanssa, on vaikeaa päätellä mihin haaraan muutokset alunperin tehtiin. Tämä onnistuu lähinnä yhdistettäessä muodostuvan muutoksen muutosviestistä (engl. commit message). Toinen vaihtoehto olisi tutkia Gitin reflog-toiminnon tekemiä tallenteita, mutta ne ovat tallennettuna vain siihen tietovarastoon, jossa muutokset tehtiin, eivätkä ne siirry työnnettäessä muutoksia toiseen tietovarastoon. Tämä on ongelma, sillä tiimi käytti uusien ominaisuuksien tekemiseen uusia haaroja, ja datanlouhinnan kannalta olisi hyödyllistä, mikäli saataisiin selvitettyä, mihin haaroihin, ja siten ominaisuuksiin, viedyt muutokset liittyvät. Näistä rajoitteista johtuen Jirasta saatu data osoittautui versionhallintadataa oleellisemmaksi.

Tehtävänhallintajärjestelmä Jira on toinen datan lähde, ja se osoittautui hyödyllisemmäksi kuin versionhallintadata. Koska Jiran tehtävät liittyvät suoraan projektissa tehtäviin tehtäviin, niiden avulla oli huomattavasti helpompi tehdä päätelmiä projektista. Jiran tietoon pääsee käsiksi esimerkiksi tekemällä suoria SQL-kyselyitä



Kuva 4.3: CVSAnalYn tietokanta. [27]

sen tietokantaan. Jiran tietokannan skeema [7] on kuitenkin varsin monimutkainen, mutta yhteistyöyhtiö hoiti tiedon hankkimisen tietokannasta SQL-kyselyillä, joten emme itse joutuneet tekemään komentosarjoja tätä varten. Data saatiin anonymisoituna csv- ja tsv-muotoisissa tiedostoissa. Tehtävät on kuvattu tsv-muotoisissa tiedostoissa, joissa on tehtävien id, tyyppi, ja ajat jolloin tehtävä on luotu, päivitetty, ratkaistu ja milloin sen määräaika on. Data sprinteistä saatiin tsv-tiedostona, jossa on tieto onko sprintti suljettu, sekä alku-, loppu- ja valmistumispäivämäärät sekä nimi. Lisäksi myöhemmin saatiin tehtäviin liittyvät tilojen muutokset, joissa on tehtävän ID, vanha tila, uusi tila ja muutoksen aika.

4.3 CVSAnalY

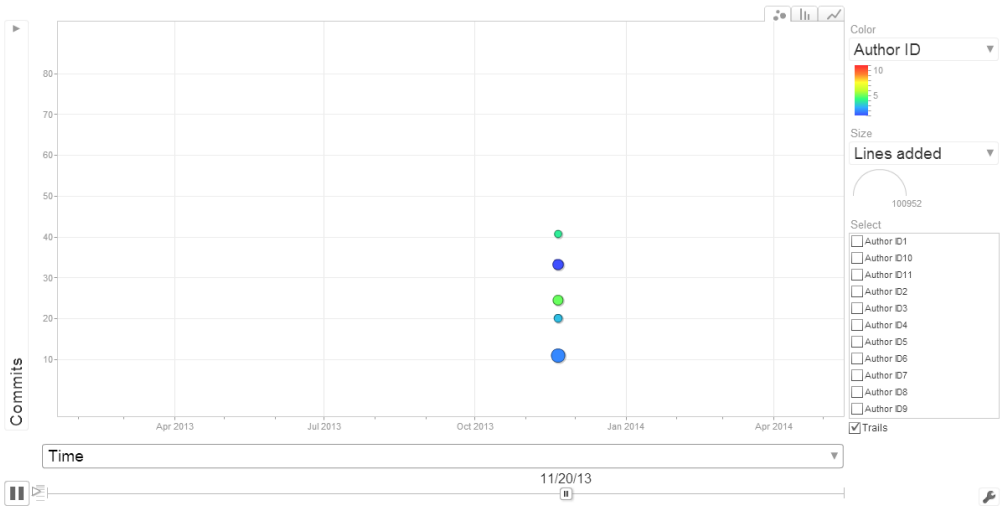
CVSAnalY on työkalu joka on tarkoitettu tiedon keräämiseen erilaisista versionhallintajärjestelmistä [14]. CVSAnalY tukee Git, Subversion ja CVS -versionhallintajärjestelmiä. CVSAnalYn toiminta voidaan jakaa kolmeen päävaiheeseen: esiprosessointi, datan tallentaminen tietokantaan ja jälkiprosessointi, mutta nämä vaiheet voidaan jakaa myös useampaan alivaiheeseen. Esiprosessointiin ja jäsennykseen kuuluu lähdekoodien lataus tutkittavasta tietovarastosta. Kun se on tehty, noudetaan lokit ja niiden sisältämä data muunnetaan strukturoidumpaan muotoon. Jäsentämisen aikana jokainen tiedosto myös jaotellaan tyypin (esim. koodi, dokumentaatio, kuvat jne.) mukaan, mikä mahdollistaa päätelmät siitä, että minkä alueen parissa kukin projektiin osallistunut henkilö on työskennellyt. Yleensä tämä tehdään tiedostopäätteen perusteella.

Varsinaisessa datan käsittelyssä ja tallentamisessa tietokantaan tietoa järjestellään järkeväpään muotoon. Aluksi data on yhdessä tietokannan taulussa, jossa on rivi jokaista muutosta kohden. Tiedon hakeminen tällaisesta raakadatasta on hankalaa, minkä takia jonkinlaista käsittelyä tarvitaan. Esimerkiksi muutoksen viejille tehdään tunnus ja tietokantaan tallennetaan tieto siitä, että mihin tiedostoihin on tehty muutoksia. Jälkiprosessointiin kuuluu erilaisten statistiikkaa keräävien lisäosien, kuten esimerkiksi Metricsin ajaminen. [33]

4.4 Käytetyt visualisointityökalut

Google Charts [20] on Googlen tekemä kirjasto, joka on tarkoitettu yksinkertaisien kuvaajien kuten pylväsdiagrammien ja viivakaavioiden piirtämiseen, ja niiden sisällyttämiseen web-sivuille. Yleensä Google Chartsia käytetään nettisivuilla olevalla JavaScript-koodilla, joka lataa kirjaston, muokkaa datan sopivaan muotoon ja asettaa kuvaajan asetukset. Google Chartsin käyttö on varsin yksinkertaista, eikä se vaadi paljoa opettelua, mutta toisaalta käyttäjä on rajoitettu olemassaoleviin kaavioihin sekä niissä oleviin asetuksiin. Projektin alussa käytimme Google Chartsia, mutta sen rajoitusten takia siirryimme käyttämään D3.js-visualisointikirjastoa. Kuvassa 4.4 on esimerkki Google Chartsin Motion Chartista. Motion Chartissa yhtenä akselina on aika, ja y-akselilla on joku muuttuja, jonka arvoa halutaan tarkastella. Lisäksi x-akselilla on joku muuttuja, mikä mahdollistaa y:n muutoksen tarkkailun sekä ajan että x:n suhteen. Lisäksi kuvaajassa olevien ympyröiden koolla voidaan ilmaista neljättä muuttujaa ja tarvittaessa värillä viidettä, joten tällaisella kaaviolla voidaan esittää varsin monta dimensiota omaavaa tietoa.

D3.js [13] on pääasiallisesti visualisointiin tarkoitettu grafiikkakirjasto JavaScriptille. Sitä käytetään esim. datan esittämiseen graafisesti nettisivuilla HTML:n (engl. Hypertext Markup Language), SVG:n (engl. Scalable Vector Graphics) ja CSS:n



Kuva 4.4: Motion chart, yksi Google Chartsin kaaviotyypeistä.

(engl. Cascading Style Sheets) avulla. D3:n avulla voi sitoa vapaavalintaista dataa sivun DOMiin (engl. Document Object Model), esimerkiksi D3:n avulla voi generoida HTML-taulukon numeroista tai tehdä samasta datasta SVG-pylväsdiagrammin.

D3 ei ole monoliittinen sovelluskehys, joka yrittää tarjota kaikki mahdolliset ominaisuudet, vaan se yrittää ratkaista ongelman ytimen eli dokumenttien tehokkaan muokkaamisen datan pohjalta. Tällä lähestymistavalla vältetään suljettujen kirjastojen ongelmia, kuten rajoittunutta muokattavuutta. D3 pyrkii minimoimaan ylimääräisen prosessorikuorman, mikä tekee siitä nopean ja mahdollistaa suuret datajoukot ja dynaamisuuden interaktiivisuutta ja animaatioita varten.

Dokumenttien muokkaaminen tavallisen DOM API:n avulla on työlästä, sillä metodien nimet ovat pitkiä ja kyseessä oleva lähestymistapa vaatii manuaalista iterointia esimerkiksi for-luupilla. Esimerkiksi kappaleiden värien muuttaminen vaatii silmukkaa. D3:lla sama onnistuu yhdellä rivillä koodia:

```
d3.selectAll("p").style("color", "white");
```

5. VISUALISOINTIKIRJASTON KEHITYS

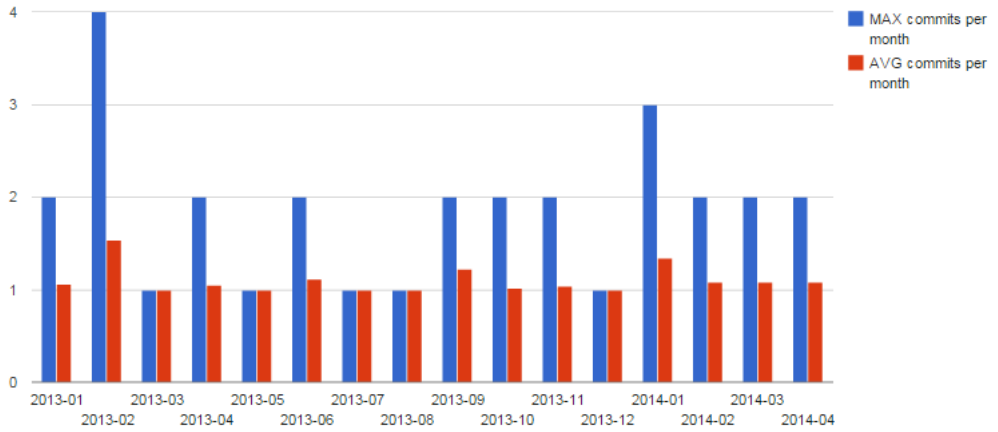
Luvussa 5.1 kerrotaan visualisointityökalun kehittämisestä, ja siitä mitä kautta päädyttiin kyseisiin ratkaisuihin. Luvussa 5.2 kuvataan visualisaatiotyökalun arkkitehtuuri, sekä se miten sitä käytetään, ja esitetään esimerkki käytöstä. Luvussa 5.3 on kuvattu seuraavan version kehityksen alkua, ja sitä, miksi tämä versio oli tietyissä asioissa puutteellinen.

5.1 Kuvaus kehityksestä

Visualisaation kehittäminen aloitettiin yksinkertaisilla Google Chartsilla [20] tehdyillä visualisaatioilla versionhallintadatasta, sillä sitä oli helppo käyttää ja sillä sai nopeasti aikaan ensimmäiset tulokset. Lisäksi se käyttää kuvien piirtämiseen Javascriptillä generoituja HTML- ja SVG-elementtjä, minkä takia sen yhteensopivuus eri selainten kanssa on hyvä. Ensimmäiset visualisaatiot olivat yksinkertaisia pylväsdiagrammeja ja viivadiagrammeja, jotka kuvasivat esimerkiksi vietyjä muutoksia kuukautta kohden tai lisättyjä tai poistettuja rivimääriä. Kuvassa 5.1 on esimerkki näistä ensimmäisistä kuvaajista. Kokeilimme myös käyttää Google Chartsin Motion Chartia, joka oli tehty Adobe Flashillä. Se kuitenkin osoittautui huonoksi käyttötarkoitukseemme visuaalisesta näyttävyydestään huolimatta, sillä sen avulla oli vaikea saada kokonaiskuvaa tutkittavasta ohjelmistoprojektista.

Yksi ensimmäisistä ideoista datan visualisointiin oli vietyjen muutosten esiintymistiheyden visualisoiminen tiedostokohtaisesti, mutta osoittautui, ettei se ollut järkevää, sillä projekteissa oli suuri määrä lähdekooditiedostoja, mutta pieni määrä muutosten vientejä tiedostoa kohden. Tulimme siitä johtopäätökseen, että pelkkä data muutosten viennistä ei ole riittävä kuvan saamiseen projekteista, joten harkitsimme muita vaihtoehtoja ja päätimme kokeilla yhdistää Jirasta saatua sprintti- ja tehtävädataa visualisaatioon samaan tapaan kuten Lehtosen tutkimuksessa [28] on tehty. Vaikka Google Charts on suhteellisen joustava ja helppokäyttöinen työkalu visualisaatioiden tekemiseen, Jirasta saadun tehtävä- ja sprinttidatan visualisoiminen samassa kuvaajassa muutoksien viennistä olevan datan kanssa olisi ollut hankalaa. Siksi päädyttiin käyttämään D3.js:ää [13] työkaluna.

Kun D3.js:ää oli kokeiltu yksinkertaisten kuvaajien, kuten normaalien pylväs- ja viivadiagrammien piirtämiseen, aloimme suunnitella tarkoitusta varten räätälöityä visualisaatiota. Visualisaation lähtökohta oli viivadiagrammi, joka kuvaa muutoksien

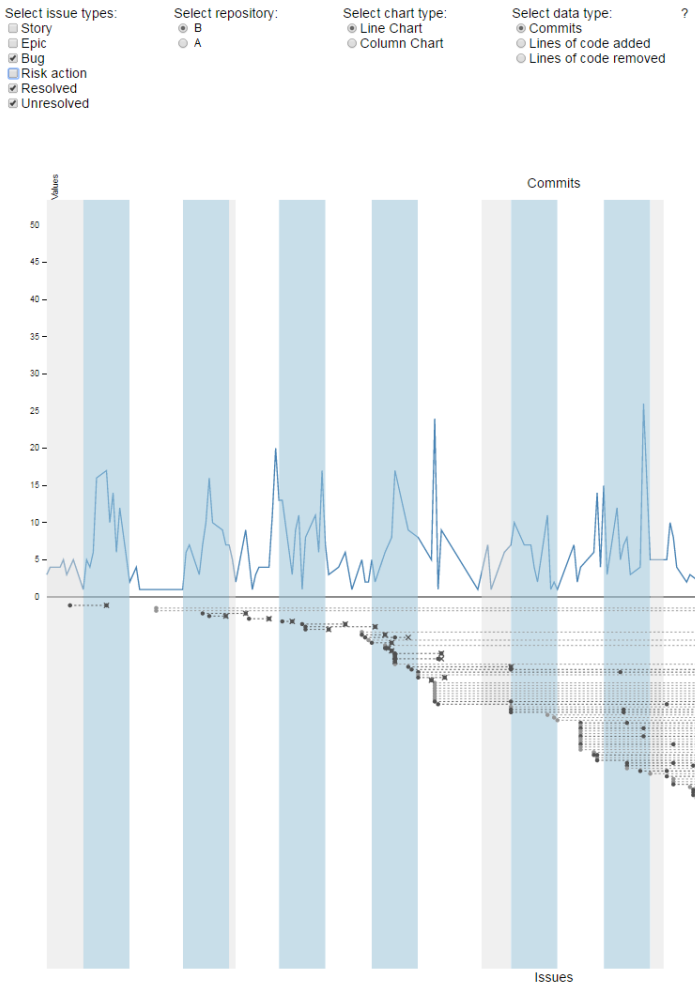


Kuva 5.1: Yksi aikaisimpia Google Chartsilla tehtyjä kuvaajia, tässä näkyy se, että tiedostoihin joihin on tehty muutoksia, on tehty keskimäärin vain hieman yli yksi muutos kuukaudessa. Myös suurin määrä muutoksia kuukauden aikana yhteen tiedostoon on varsin pieni, eli viedyt muutokset ovat melko hyvin hajaantuneet eri tiedostoihin.

vientiä päivää kohden, ja jonka taustalla on sprinttejä kuvaava aikajana; sprinttien alku- ja loppuajat on kuvattu vaaleansinisillä ja valkoisilla alueilla. Tehtäviä kuvattiin viivoilla, joiden päissä oli ympyrät. Lisäksi HTML-sivulla oli suodattimia näytettävien tehtävätyyppien valitsemiseksi, sekä visualisaatiota pystyi suurentamaan aika-akselin suhteen jotta jotain tiettyä aikaväliä pystyisi tutkimaan lähemmin. Tuloksena saatu visualisaation versio 1 on nähtävissä kuvassa 5.2. Muutoksien vienti (engl. commit) per päivä on ilmaistu viivadiagrammilla, ja Jirasta saatu tehtävädata on kuvaajan alaosassa. Tehtävän luominen ja ratkaiseminen on ilmaistu palloilla, ja väli on merkitty katkoviivalla. Jirasta saatu sprinttidata taas näkyy taustalla valkoisina ja vaaleansinisinä alueina, jotka kuvaavat sprinttien alkua ja loppua, ja visualisaatioon toteutettiin mahdollisuus suodattaa dataa HTML-elementeillä tehtävän valinnan avulla. Visualisaation suunnittelun apuna käytettiin Tuften kirjassa esiteltyjä periaatteita, jotka on käyty tarkemmin läpi luvussa 3. Lyhyesti ne ovat seuraavat:

- Kuvaajan ei tulisi valehdella tai vääristää dataa. Esimerkiksi palkin aloittaminen muualta kuin nollasta yleensä vääristää dataa. Pinta-alan tulisi olla suoraan verrannollinen datassa oleviin numeroihin, ja mittakaavan tulisi olla sama koko kuvaajassa.

- Ennen kaikkea tulee näyttää data. Koristelut ja muu “muste”, mitä ei ole käytetty datan ilmaisemiseen, pienentää datan ilmaisemiseen käytetyn musteen osuutta. Esimerkiksi ruudukko ei ole datan ilmaisemiseen kuluvaa mustetta.

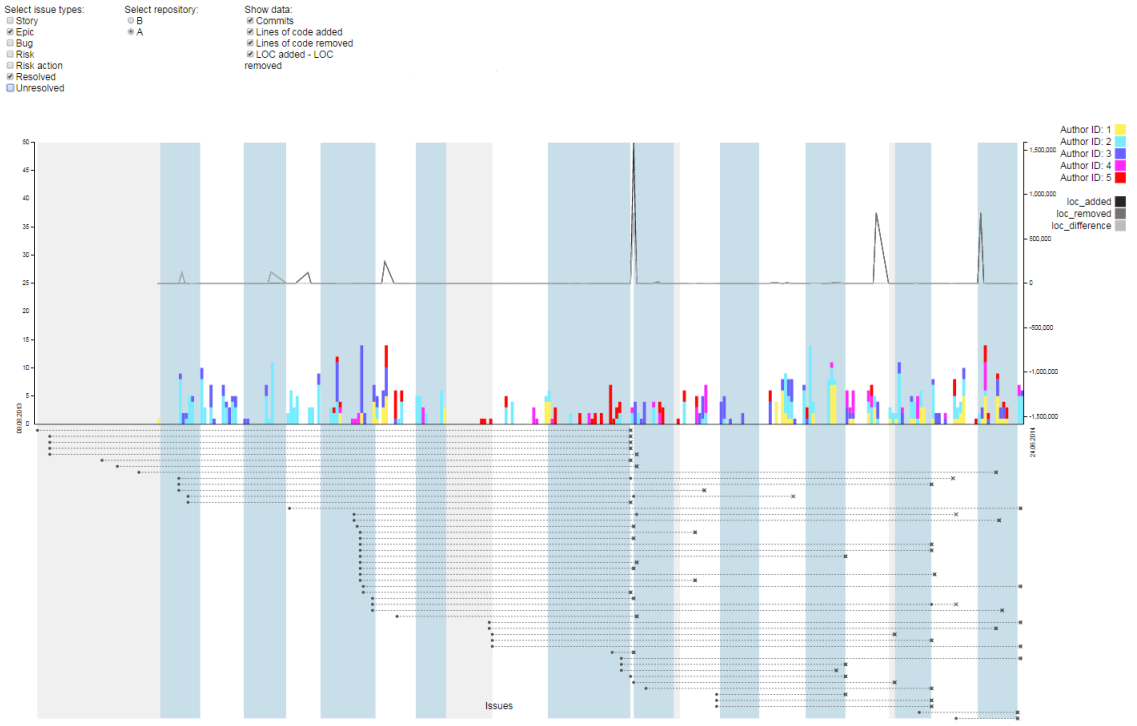


Kuva 5.2: Visualisaation versio 1.

- Vältä kaaviojätettä. Esimerkiksi erityisen tumma tai paksu ruudukko on kaaviojätettä, joka vaikeuttaa datan näkemistä. Yleensä harmaa ruudukko on parempi kuin musta.

Nämä periaatteet näkyivät visualisaatiossa pääasiassa siten että se on melko pelkistetty, turhia koristeluja tai ruudukoita on vältetty. Aluksi vietyjen muutosten ja rivimäärien mittakaava riippui näytettävästä tietovarastosta, minkä takia mittakaava oli eri tietovarastojen välillä. Tämä aiheuttikin visualisaation tulkinnassa väärinkäsityksiä, mistä nähdään että kyseisiä periaatteita on hyvä miettiä, vaikka ne kuulostavatkin itsestäänselvyyksiltä.

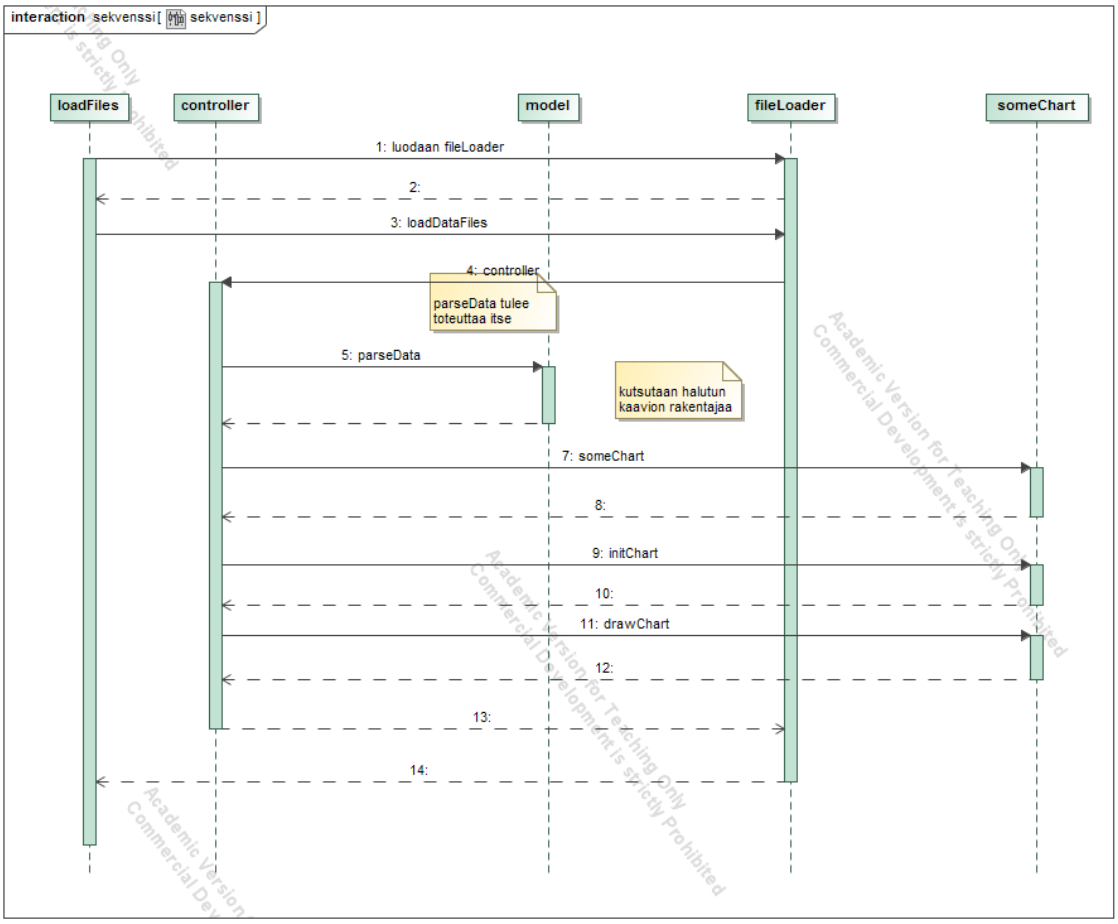
Datan yhdistämisestä eri lähteistä ja projektien esittämisestä samanlaisella aikajanalla saatiin hyvää palautetta yhteistyöyhtiöltä, ja tämä oli hyödyllisin askel visualisaation kehittämisessä. Se paransi analysointia huomattavasti, sillä se paljasti paremmin korrelaatioita tai anomalioita datajoukkojen välillä. Seuraavaksi visualisaatioon lisättiin dataa, kuten lisättyjen ja poistettujen rivien määrä, sekä näiden



Kuva 5.3: Visualisaation versio 2. Viedyt muutokset per päivä on ilmaistu pylväsdiagrammilla, minkä lisäksi eri kehittäjien tekemät muutokset on visualisoitu värien avulla. Muutosten lisäämät ja poistamat rivimäärät on piirretty päiväkohtaisesti viivadiagrammilla.

erotus. Lisäksi vietyjen muutosten määrä näytettiin kehittäjäkohtaisesti kumulatiivisena pylväsdiagrammina tai viivadiagrammina. Viivadiagrammi osoittautui tässä kuitenkin melko epäselväksi, sillä kehittäjiä oli useita ja viivoja oli vaikeaa erottaa toisistaan. Tämän takia tultiin tulokseen, että pylväsdiagrammi sopii paremmin kehittäjäkohtaisten muutosmäärien näyttämiseen. Versio 2 visualisaatiosta on nähtävissä kuvassa 5.3, ja tätä versiota käytettiin analyysin apuna artikkelissa What Can be Revealed by Visualizing Software Engineering Data? - a Case Study [29] kuvatussa tutkimuksessa.

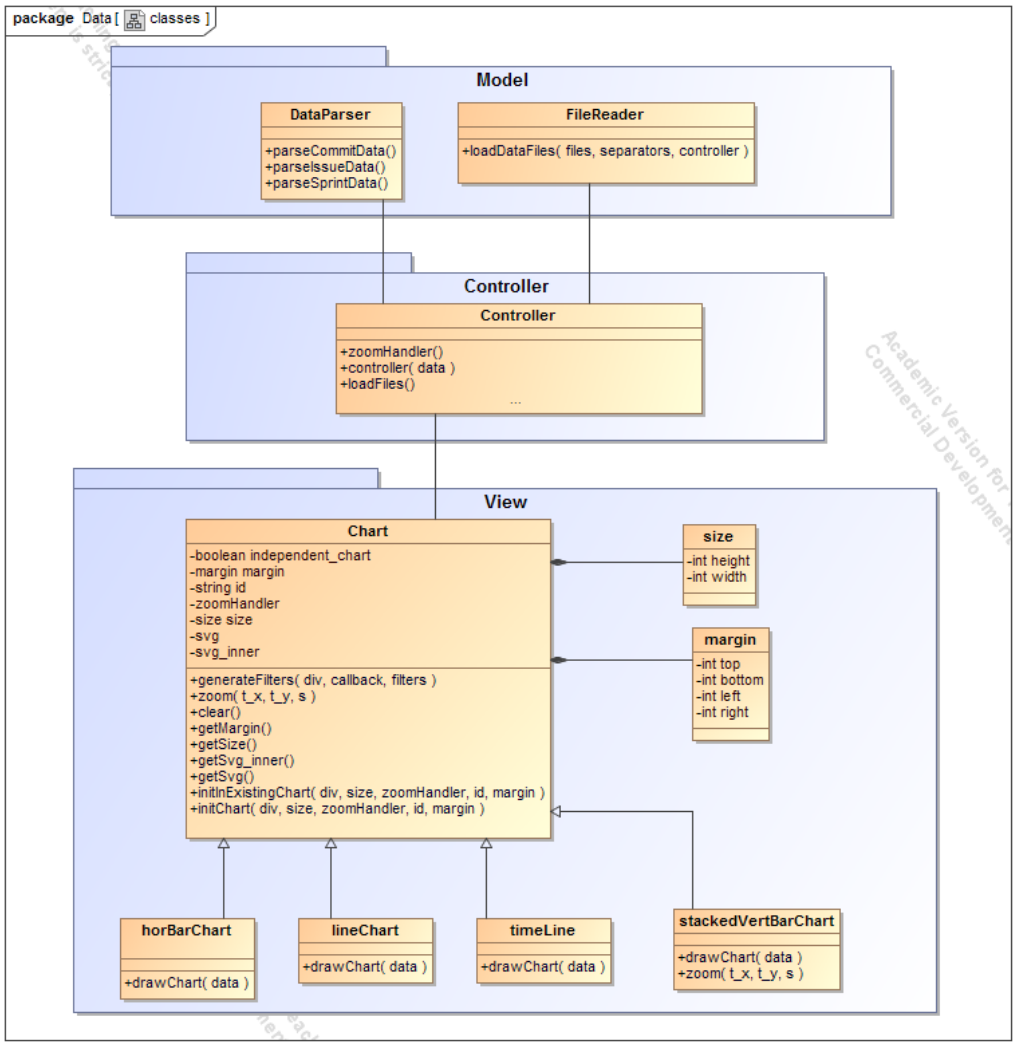
Visualisaation versio 2 soveltui kyseisen tutkimuksen apuvälineeksi, mutta sen muokattavuus oli huono. Tämä johtui siitä, että siihen oli vain lisätty ominaisuuksia arkkitehtuuria suuremmin miettimättä, minkä takia tuloksena oli “spagettikoodia”. Sen takia päätettiin refaktoroida koodi, jotta siitä tulisi uudelleenkäytettävämpi. Tarkoituksena oli myös tehdä yleisemmin Need for Speed -projektiin sopiva työkalu, jonka avulla voisi suhteellisen pienellä vaivalla tehdä visualisaatioita. Refaktorointi aloitettiin yksinkertaisesti siirtämällä osia toiminnallisuudesta eri tiedostoihin. Eri kaavioitten piirtämiseen käytettävä koodi siirrettiin omiin kooditiedostoihinsa, ja samoin tiedostojen lukemiseen ja jäsentämiseen käytettävä koodi siirrettiin omiin kooditiedostoihinsa. Kaavioiden alustaminen tehtiin pääkooditiedostossa, ja kaaviot piirtävät kooditiedostot sisältivät myös toiminnallisuuden suodattimien piir-



Kuva 5.4: Visualisaation version 4 sekvenssikaavio.

tämiseen ja datan uudelleenpiirtämiseen suodattimien pohjalta. Lisäksi kaavioissa oli mahdollisuus piirtää ne olemassaolevien kaavioiden päälle, mikä oli tarpeellista käyttämämme visualisaation kaltaisen kuvaajan tuottamiseen.

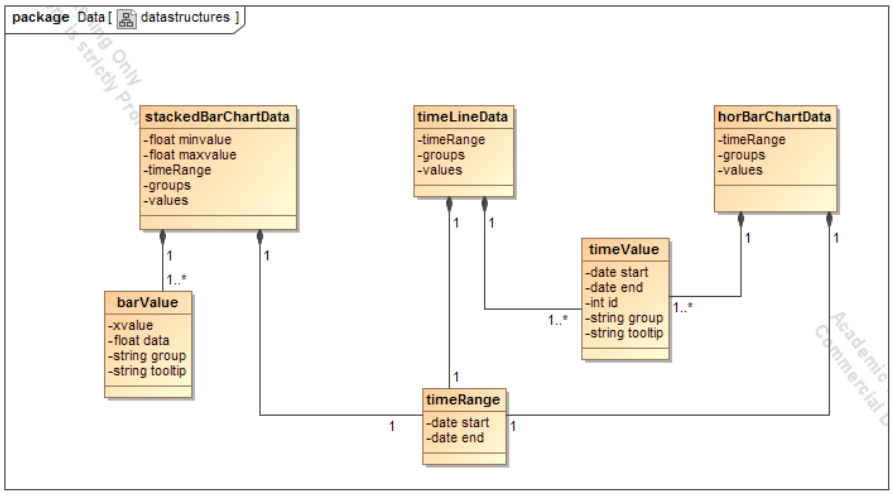
Vaikka tuloksena saatu versio 3 olikin parempi kuin alun “spagettikoodilla” tehty versio, se ei silti osoittautunut riittävän helppokäyttöiseksi uusien visualisaatioiden tekemiseen. Tämä ilmeni erään toisen, projektiin kuuluvan henkilön yrittäessä tehdä visualisaatiota, ja tähän menikin parin tunnin sijaan noin 10 tuntia, jona aikana työkalusta löytyi useita arkkitehtuuriin liittyviä ongelmia sekä virheitä. Pääasiallisena ongelmana oli se, että visualisaation muuttaminen vaati useita muutoksia eri osissa ohjelmakoodia. Esimerkiksi tiedoston lukeminen ja jäsentäminen täytyi muokata raakadataan sopivaksi, suodattimien tekemistä kaavioiden piirtämisen sisältävissä kooditiedostoissa piti muuttaa halutunlaiseksi ja lisäksi kaavioiden alustaminen ja datan antaminen niille piti toteuttaa uudestaan. Lyhyesti sanottuna useimpiin käytettyihin kooditiedostoihin joutui tekemään muutoksia. Tämän lisäksi Javascriptin omat erityispiirteet vaikeuttivat virheiden löytämistä. Esimerkiksi Javascriptissä



Kuva 5.5: Visualisaation version 4 luokkakaavio.

muuttujasta tulee globaali, jos sen esittelyssä ei ole edessä varattua sanaa var. Tämä aiheutti useita vaikeasti selitettävään ohjelman toimintaan johtaneita virheitä, joita oli vaikea löytää, sillä niistä ei tullut minkäänlaista virheilmoitusta.

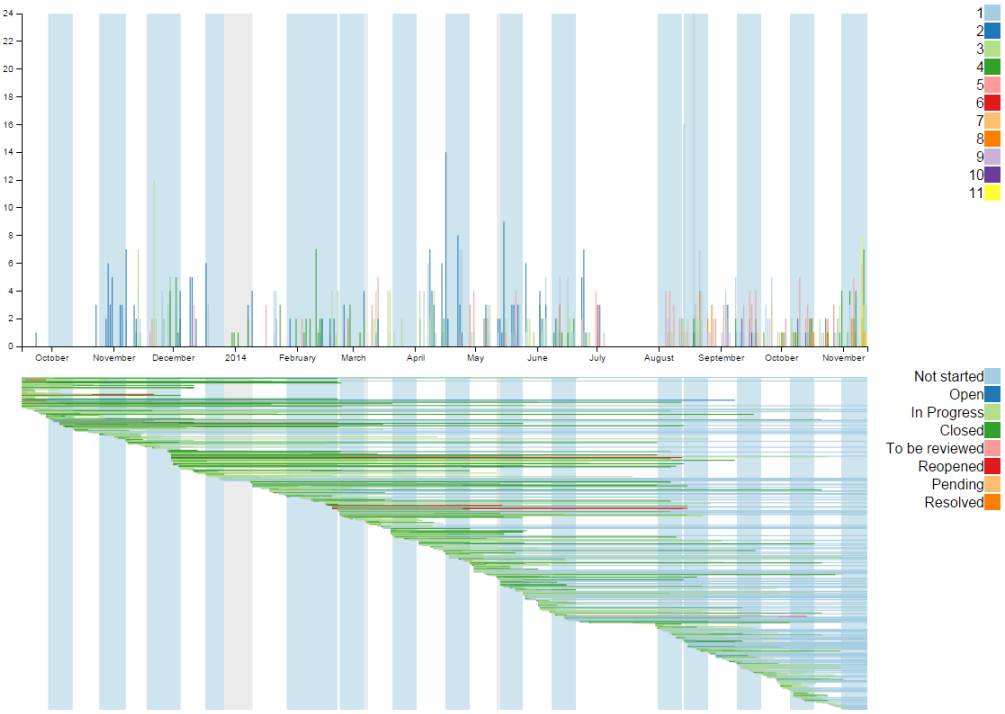
Versiosta 4 oli tarkoitus tehdä tätä versiota yleiskäyttöisempi ja helppokäyttöisempi, ja sen tekemiseen osallistui myös toinen työntekijä. Yhtenä keinona uudelleenkäytettävyyteen pyrkimisessä päätimme käyttää MVC-arkkitehtuuria [4], jotta toiminnallisuus saataisiin pilkottua erillisiin osiin. MVC-mallin ideana on, että järjestelmä on jaettu malliin, näkymään ja ohjaimeen. Malli kuvaa järjestelmän tiedon tallentamisen sekä käsittelyn, esimerkiksi tietokannan käsittely on toteutettu mallissa. Näkymä määrittää käyttöliittymän ja ulkoasun, sekä se esittää tiedot. Ohjain taas vastaanottaa käyttäjältä tulevat käskyt, sekä muuttaa mallissa ja näkymässä olevia tietoja käskyjen mukaisesti. Visualisointityökalun tapauksessa datan lukeminen ja jäsentäminen oli tarkoitus tehdä mallissa, kaavioiden alustaminen ja käyttä-



Kuva 5.6: Visualisaation version 4 kaavioiden käyttämät tietorakenteet.

minen ohjaimessa, sekä itse kaaviot ja suodattimet näkymässä. MVC-mallin käyttämisessä työkaluun oli joitain ongelmia. Ensinnäkin työkalun toiminta on luonteeltaan hyvin liukuhihnamaista. Ensin data luetaan jostain, esimerkiksi csv-tiedostoista, sitten se jäsennetään ja käsitellään piirtokomponenteille sopivaan muotoon, minkä jälkeen se annetaan piirrettäville komponenteille. Ainoastaan suodattimet ja suurentamismahdollisuus antavat käyttäjän vaikuttaa ohjelman toimintaan millään tavalla, eikä näihin toimintoihin liittyviä tietoja tallenneta mihinkään.

Aluksi malliin toteutettavasta tiedostojen lukemisesta ja jäsentämisestä piti tulla yleiskäyttöinen, jossa mallille annettaisiin csv-tiedostojen nimet, niistä haluttujen sarakkeiden nimet sekä suodattimet joilla valitaan datasta haluttu osajoukko. Lisäksi alussa kaavailtiin myös velhoa, jonka avulla datasta voitaisiin yhdistää halutut tiedot kuvaajassa vastaaviin ominaisuuksiin, kuten y:n arvoon tai aikaväliin. Tällaisen yleiskäyttöisen tiedostojen lukemisen ja jäsentämisen toteuttaminen osoittautui kuitenkin odotettua huomattavasti vaikeammaksi. Pääasiallinen syy tähän oli se, että käytetyllä datalla ei ollut yhtenäistä formaattia, ja se oli hajaantunut useisiin tiedostoihin. Esimerkiksi Jiran tehtävistä saatu data on kahdessa erillisessä tiedostossa. Yhdessä tiedostossa on itse tehtävät ja niiden tiedot, kuten luomispäivämäärä, id, nimi, tehtävän tyyppi ja sulkemispäivämäärä. Tehtävien tilojen muutokset ovat taas erillisessä tiedostossa, jossa on tilanmuutokseen liittyvän tehtävän id, tilan muutoksen päivämäärä, edellinen tila ja uusi tila. Tällaisen useassa tiedostossa olevan datan jäsentämisen toteuttaminen geneerisesti olisi hyvin haastavaa, ja käytännössä vaatisi tätä varten toteutetun käyttöliittymän. Näin suuren järjestelmän toteuttaminen kahden ihmisen voimin muutamassa kuukaudessa ei ollut realistista, ja yritettyämme tuloksetta tehdä yleiskäyttöistä datan lukijaa ja jäsentäjää, päätimme jättää datan jäsentämisen käyttäjän vastuulle.



Kuva 5.7: Esimerkki työkalulla tehdystä visualisaatiosta.

Version 4 “luokkakaavio” on kuvassa 5.5. Javascriptissä ei ole luokkia, vaan se käyttää prototypaalista perintää, minkä takia kaaviossa olevat luokat ovat paremminkin funktioita, joihin kyseiset osat koodia on sijoitettu. Näkymässä yhteisiä ominaisuuksia sisältävästä perusluokasta Chart on periytetty erilaiset kaaviotyypit, jotka toteuttavat itse kaavion piirron ja mahdollisesti toteuttavat uudestaan joitain ominaisuuksia, kuten suurentamisen. Malli sisältää käytännössä vain tiedostojen lukemisen ja jäsentämisen. Kuvassa 5.4 olevassa sekvenssikaaviossa on esimerkki visualisoinnin tekemisestä. Kun html-sivu on ladattu, kutsutaan funktiota loadFiles, jossa taas kutsutaan funktiota fileLoader. Parametrinä annetaan taulukoissa halutut tiedostonimet, sekä datassa olevat erotinmerkit, kuten pilkku tai puolipiste. Ladattuaan tiedostot fileLoader kutsuu ohjainta, jonka vastuulla on kutsua kaavioille sopivan tietorakenteen tuottavaa datan jäsentäjää, sekä alustaa kaaviot oikeisiin kohtiin sivua. Lopuksi ohjain antaa kaavioille datan ja käskee niitä piirtämään kuvaajat. Kaavioiden alustaminen olemassaolevan kaavion päälle on myös mahdollista, tosin itse piirtäminen tapahtuu siinä järjestyksessä, missä kaavioiden piirtoa kutsutaan. Piirtojärjestys vaikuttaa siihen, minkä kaavion elementit ovat päällimmäisenä. Tämä johtuu siitä, että SVG-elementeille ei voi määritellä kerrosta, vaan ne tulevat päällekäin piirtojärjestyksessä, ensin piirretyt ovat alimmaisena.

```
<!DOCTYPE html PUBLIC>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <style>
    .axis text {
      font: 10px sans-serif;
    }

    .axis path,
    .axis line {
      fill: none;
      stroke: #000;
      shape-rendering: crispEdges;
    }
  </style>

  <meta http-equiv="content-type" content="text/html; charset=utf-8"/>

  <title>Visualizing data</title>
  <script type="text/javascript" src="lib/d3.js"></script>
  <script type="text/javascript" src="lib/map.js"></script>
  <script type="text/javascript" src="lib/jquery-1.11.1.js"></script>

  <script type="text/javascript" src="view/stackedvertbarchart.js"></script>
  <script type="text/javascript" src="view/timeline.js"></script>
  <script type="text/javascript" src="view/horbarchart.js"></script>
  <script type="text/javascript" src="view/chart.js"></script>
  <script type="text/javascript" src="controller/controller.js"></script>
  <script type="text/javascript" src="model/parser.js"></script>
  <script type="text/javascript" src="model/filereader.js"></script>

  </head>
  <body onload="loadFiles()" style="font-family: Arial; border: 0 none;">
    <div id="chart1" style="float:left;"></div>
    <div id="chart2" style="float:left;"></div>
  </body>
</html>
```

Listaus 1: Listaus html-sivusta jolle kaaviot asetetaan.

5.2 Esimerkki kirjaston käytöstä

Kirjaston käyttöä varten tulee sivulle luoda sopivat DOM-elementit, joihin kaaviot piirretään. Lisäksi sivulle tulisi lisätä bodyOnLoad -funktioksi esimerkiksi loadFiles, jossa ladataan visualisoinnissa käytettävät csv-muotoiset tiedostot, ja luonnollisesti visualisaatiossa käytettävät javascript-tiedostot tulee lisätä sivulla. Listauksessa 2 on esimerkki funktiosta loadFiles, jossa luodaan tiedostojen lukija, asetetaan taulukkoon luettavien tiedostojen nimet ja polut, sekä toiseen taulukkoon erotinmerkit. Tabulaattori tulee kirjoittaa \t avulla. Lopuksi kutsutaan lukijan funktiota loadDataFiles, jolle annetaan parametriksi taulukot tiedostojen nimistä ja erotin-

```
function loadFiles() {
    var filereader_ = new fileReader();
    var files = ["data/commits.csv",
                 "data/issues.tsv",
                 "data/sprints.csv",
                 "data/statuschanges.tsv"];
    var separators = [",","\t", ",", "\t"];
    filereader_.loadDataFiles(files, separators, controller);
}
```

Listaus 2: Funktio loadFiles.

merkeistä, sekä ohjain, jota kutsutaan tiedostojen lataamisen jälkeen.

Ohjaimelle (listaus 3) annetaan parametriksi tiedostonlukijan tekemä tietorakenne. Sen tehtävänä on luoda kaaviot ja alustaa ne haluttuihin kohtiin sivua, sekä kutsua jäsentäjiä ja lopulta piirtää kuvaajat. Kaavioiden luominen tapahtuu riveillä 3-6, ja kohde-elementtien valinta tapahtuu d3:n valmiilla funktioilla riveillä 20-21. Riveillä 26 ja 27 alustetaan sprinttejä kuvaavat timeline-kaaviot kyseisiin DOM-elementteihin. Elementin lisäksi parametriksi annetaan kaavion haluttu koko, funktio zoomHandler, kaavion ID ja marginaalit, jotka halutaan kaavion ympärille. Funktiota zoomHandler käytetään sen takia, koska esimerkissä halutaan, että kaavioita suurennettaessa muutkin kaaviot suurentuvat. Tämän takia suurentamisen käsittelijä tulee toteuttaa itse. Tässä tapauksessa se ottaa huomioon vain x-akselin suuntaisen suurentamisen, ja kutsuu kaavioiden funktiota zoom, jolle annetaan parametriksi translaatio x- ja y-akselin suunnassa, sekä mittakaava. Kun timeline-kaaviot on alustettu tällä tavalla, alustetaan pylväsdiagrammi ja kuvaaja Jiran tehtävistä niiden päälle riveillä 28-29. Koska kuvaaja alustetaan toisen päälle, ei tarvita osaa parametreistä kuten kokoa tai DOM-elementtiä johon se laitetaan. Siksi pelkästään kaavio, jonka päälle se piirretään, zoomHandleri ja ID riittävät.

Ennen kaavion piirtämistä tiedonlukijalta saatu data tulee jäsentää kaavioille kelpaavaan formaattiin, joka on kuvattu kuvassa 5.6. Lisäksi rivillä 36 asetetaan piirrettävä aikaväli samaksi kuin toisessa kuvaajassa, jotta molempien kuvaajien aika-akseli olisi sama. Lopuksi kaaviot piirretään, piirrettäessä on syytä huomioida, että SVG-kuvaajien luonteen takia kaaviot piirretään päällekäin siinä järjestyksessä kuin piirtofunktioita kutsutaan. Loput kooditiedostot ovat liitteessä A, eikä niitä käydä tässä tarkemmin läpi.

5.3 Jatkokehitys

Suunnitelmista poiketen versioon 4 ei ehditty toteuttaa suodattimia, sillä tässäkin lähestymistavassa huomattiin puutteita. Vaikka itse kaaviot olivatkin uudelleenkäytettäviä, datan kerääminen ja jäsentämisen toteuttaminen piti edelleen toteut-

```
function controller(data) {
    //luodaan kaavio-oliot
    var stackedcolumnchart = stackedVertBarChart();
    var sprints1 = timeLine();
    var sprints2 = timeLine();
    var issuechart = horBarChart();

    //Zoom handler. Tätä kutsutaan kaavioista, jotta usean kaavion
    //samanaikainen suurentaminen olisi mahdollista.
    function zoomHandler() {
        var t = d3.event.translate[0];
        var s = d3.event.scale;
        stackedcolumnchart.zoom(t, 0, s);
        sprints1.zoom(t, 0, s);
        sprints2.zoom(t, 0, s);
        issuechart.zoom(t, 0, s);
    }

    //valitaan kaavioiden paikat ja määritellään koko ja marginaali
    var div_commits = d3.select("#chart1");
    var div_issues = d3.select("#chart2");
    var size = {width: 1200, height: 400};
    var margin = {top: 10, left: 30, right: 230, bottom: 20};

    //alustetaan kaaviot, jälkimmäiset ensimmäisten päälle
    sprints1.initChart(div_commits, size, zoomHandler, "sprints1", margin);
    sprints2.initChart(div_issues, size, zoomHandler, "sprints2", margin);
    stackedcolumnchart.initInExistingChart(sprints1, zoomHandler, "commits");
    issuechart.initInExistingChart(sprints2, zoomHandler, "issues");

    //jäsennetään datat, koska dataa voi olla montaa eri tyyppiä,
    //tulee jäsentäjä toteuttaa itse
    parser = Parser();
    sprintdata = parser.parseSprints(data[2]);
    issuedata = parser.parseIssues(data[1], data[3]);
    issuedata.timerange = sprintdata.timerange; //asetetaan aikaväli

    //piirretään kaaviot
    sprints1.drawChart(sprintdata);
    sprints2.drawChart(sprintdata);
    stackedcolumnchart.drawChart(parser.parseCommits(data[0]));
    issuechart.drawChart(issuedata);
}
```

Listaus 3: Listaus ohjaimesta.

taa itse. Esimerkiksi datan keräys ja lukeminen csv-tiedostoihin tapahtui käsin tai sekalaisella kokoelmalla Python-komentosarjoja. Tämä lisäsi uuden datan käyttöönottamiseen menevää aikaa niin paljon että työkalu oli käyttötarkoitukseensa melko epäkäytännöllinen. Esimerkiksi yhteinen tietokanta datalle voisi huomattavasti nopeuttaa työkalun käyttöä, minkä tajusimme vasta pitkään hakattuamme päätämme seinään. Huomasimme myös että projektin sisäinen kommunikaatio oli erittäin huo-

noa, esimerkiksi edellisellä kehitysaskeleella oli toivottu uudelleenkäytettävyyttä tätä tarkemmin määrittelemättä, vaikka todellisuudessa tarkoituksena oli pääasiassa vain visualisoida Jira-dataa yhdessä versionhallinnan muutosten viennistä kertovan datan kanssa.

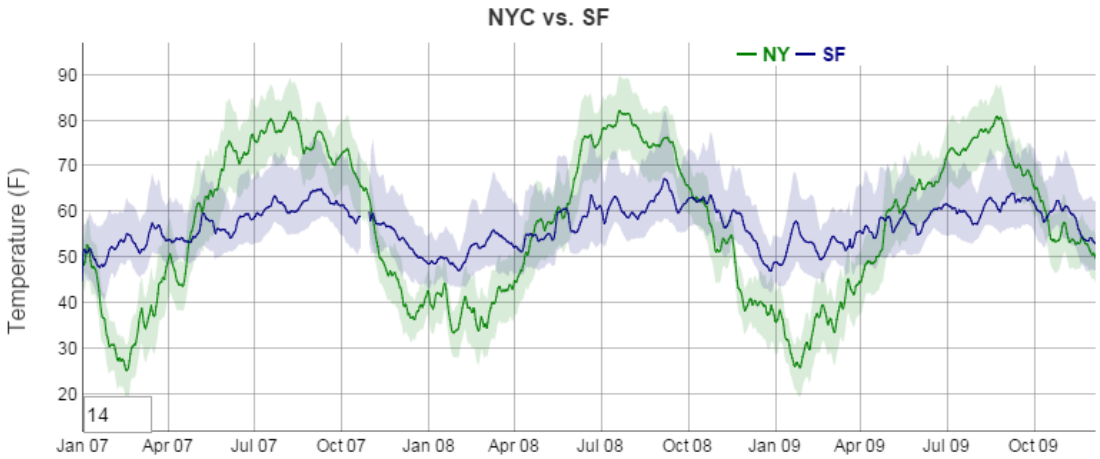
Edellä mainittujen syiden takia päätettiin alkaa kehittää uutta versiota, jossa pääasiallisena ideana on keskittää datan kerääminen tietokantaan, josta se haetaan piirrettäväksi. Tällä on tarkoitus välttää tarve ajaa erilaisia komentosarjoja, joilla data lisätään tietokantaan, tai luetaan sieltä csv-tiedostoihin. Lisäksi tällöin csv-tiedostoja ei tarvitse käsitellä selaimessa Javascriptillä, vaan tieto voidaan lukea tietokannasta suoraan valmiiksi käsitellyssä muodossa.

6. VERTAILU GENEERISIIN TYÖKALUIHIN

Datan visualisointiin on nykyisin monenlaisia työkaluja, valmiista drag & drop -periaatteella toimivista ohjelmistopaketeista sovelluskehyksiin, joiden tehtävänä on vain helpottaa oman visualisaation ohjelmointia. Esimerkiksi SAS on suuryrityskäyttöön tarkoitettu ohjelmistopaketti, joka sisältää datan visualisointiin tarkoitettujen työkalujen lisäksi välineitä datan hakemiseen, louhintaan ja hallintaan [23]. Myös tavallisten toimisto-ohjelmistojen taulukkolaskentaohjelmat, kuten Microsoft Officen Excel sisältää työkalut yksinkertaisten kaavioiden tekemiseen. Lisäksi monet tieteelliseen tai tekniseen laskentaan tarkoitetut ohjelmistot, kuten Matlab sisältävät mahdollisuuksia piirtää dataa. Tässä luvussa käydään lyhyesti läpi muutamaa työkalua, ja niiden ominaisuuksia.

Microsoft Officen Excel on perinteinen taulukkolaskentaohjelma, jossa on mahdollisuus tavallisimpien kaaviotyyppien piirtämiseen. Kaaviotyyppejä on melko paljon, mutta niiden muokattavuus on suhteellisen rajallinen, ja sen takia Excelin käyttäminen vaativampaan datan visualisointiin on hankalahkoa. Lisäksi Excelin tapauksessa käyttäjä on sidottu kyseiseen tuotteeseen, ja visualisaation siirtäminen toiseen työkaluun on hankalaa, mikä toki koskee muitakin valmiiksi tuotteistettuja kaupallisia ratkaisuja. Matlab ja muut tieteelliseen laskentaan tarkoitetut ohjelmat taas sisältävät kattavia työkaluja datan laskemiseen, sekä kuvaajien piirtämiseen tästä datasta. Ne kuitenkin on tarkoitettu enemmän jatkuvien funktioiden visualisoimiseen kuin informaation visualisoimeen, jota tässä projektissa tarvittiin.

Webbiohjelmoinnin suosion lisääntymisen myötä selainpohjaisten työkalujen määrä on lisääntynyt rajusti. Esimerkiksi Google Charts, joka on esitelty tarkemmin luvussa 4, on tällainen. Vaikka sen käyttö vaatii Javascriptin osaamista, se on silti varsin yksinkertainen käyttää, sillä se sisältää valmiita kaavioita, joille vain annetaan data. Helppokäyttöisyyden huonona puolena on kuitenkin huonompi muokattavuus, sillä käyttäjä on rajoitettu näihin kaaviotyyppeihin ja valmiiksi tehtyihin vaihtoehtoihin. Google Chartsin lisäksi esimerkiksi Timeline [31], jolla voi toteuttaa timeline-tyylisiä visualisaatioita, ja Dygraphs [1], joka sopii viivadiagrammien piirtämiseen, ovat Javascript-kirjastoja, jotka tarjoavat valmiiksi toteutettuja kaavioita. Kuvassa 6.1 on esimerkki Dygraphsilla piirretystä kuvasta, josta nähdään, että työkalu sopii tarkoitukseen hyvin. Diagrammi on selkeä, sen toteuttamiseen on kulunut vain muutama koodirivi, ja siinä on jopa ilmaistu virhemarginaalit. Toi-

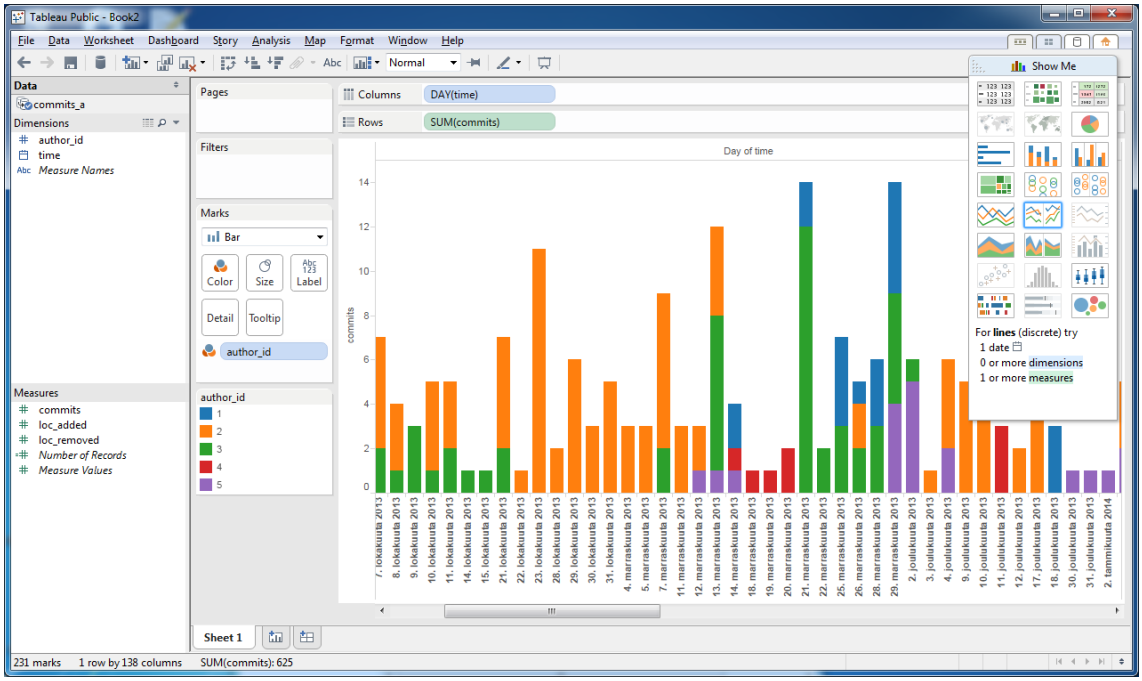


Kuva 6.1: Esimerkki Dygraphsilla tehdystä kaaviosta. Kaavio on peräisin Dygraphsin kotisivulta [1].

saalta Dygraphsilla ei kuitenkaan voi tehdä muita kaavioita kuin viivadiagrammeja, joten työkalun soveltuvuus riippuu kyseisen kaaviotyypin soveltuvuudesta käyttötarkoitukseen. On myös olemassa Javascript-kirjastoja, joiden ei ole tarkoitus tarjota valmiita, mahdollisimman helposti käytettäviä kaavioita, vaan joiden tarkoitus on helpottaa omien kaavioiden piirtämistä. Kuten luvussa 4.4 kerrottiin pääasiassa D3.js:ään liittyen, tavallisen DOM API:n kautta selaimessa näkyvien HTML- tai SVG-elementtien luominen tai muokkaaminen on varsin työlästä. Siksi nämä kirjastot pyrkivät tarjoamaan rajapinnan, jonka kautta on helpompaa luoda kaavioita peruselementeistä. D3.js:n lisäksi esimerkiksi Raphael [9] on tällainen kirjasto.

On myös olemassa D3:n päälle toteutettuja kaavioiden piirtokirjastoja, kuten NVD3 [32]. Se tarjoaa uudelleenkäytettäviä kaavioita, joille vain annetaan data Javascript-oliossa, jonka perusteella ne piirtävät kaavion. Tällaisilla kirjastoilla kaavioiden piirtäminen on helppoa, mutta mukautettujen kaavioiden tekeminen ei ole helpompaa kuin pelkän D3.js:n avulla. Siinä mielessä ne ovat käyttäjälle samanlaisia kuin itsenäiset visualisointikirjastot. Työssä tehty visualisointikirjasto on myös tällainen kirjasto, vaikkakaan kokeiluna ei samalla tavalla loppuun hiottu. Kirjastossa on lisäksi huomioitu nimenomaan tässä projektissa tarvittuja asioita, kuten tuki päällekäin piirretyille kaavioille, ja usean kuvaajan samanaikainen suurentaminen tai liikuttelu. Useimmissa valmiissa kirjastoissa ei ole tukea näille asioille. Myös kaaviotyypit on suunniteltu projektissa aiemmin tehtyjen visualisaatioiden pohjalta.

D3:a vastaavia kaavioiden ohjelmoimisen helpottamiseen pyrkiviä kirjastoja on myös perinteisemmille ohjelmointikielille. Esimerkiksi Prefuse [3] on Javalla toteutettu visualisointien tekemiseen tarkoitettu kirjasto. Se eroaa toteutukseltaan ja käytöltään melko paljon selainpohjaisista ratkaisuista, sillä se ei pohjaudu standardoituihin HTML- tai SVG-elementteihin, mutta periaate on sama: tarkoituksena ei ole



Kuva 6.2: Tableaun käyttöliittymä jossa muokataan kaaviota.

toteuttaa kattavaa visualisointiohjelmaa, vaan helpottaa omien visualisointien tekemistä.

Datan visualisointiin on myös valmiita työpöytäsovelluksia, joissa kaaviotyyppi, datalähde ja datan tyyppi valitaan graafisen käyttöliittymän kautta. Esimerkiksi Tableau on tällainen sovellus [34], ja siitä on myös olemassa ilmainen versio Tableau Public. Tableauta käytettäessa ensin tuodaan data jostain lähteestä, esimerkiksi Excel-taulukkotiedostosta tai tavallisesta csv-muotoisesta tekstitiedostosta. CSV-muotoiselle tiedostolle voi määrittää erotinmerkin, esimerkiksi tabulaattorin tai puolipisteen, ja sen perusteella Tableau tunnistaa sarakkeet automaattisesti, ja ehdottaa niille tyyppiä, kuten päivämäärä, kokonaisluku, desimaaliluku, merkkijono jne. joita voi tarvittaessa muuttaa. Itse kaavion luomiseen käytettävä ikkuna ja esimerkki kaaviosta näkyy kuvassa 6.2. Piirrettäessä kaaviota, niistä valitaan haluttu määrä dimensioita ja arvoja, sekä valitaan haluttu kaaviotyyppi. Valittavana on useimmat tavalliset kuvaajatyypit, kuten viivadiagrammit, pylväsdiagrammit, ympyrädiagrammit, sekä esimerkiksi dataa kartalle visualisoivia kaaviotyyppejä. Tämän jälkeen akseleiden asetuksia voi muuttaa. Esimerkiksi aika-akselin tapauksessa voi päättää, käsitelläänkö aikaa diskreettinä muuttujana, esim. ryhmitelläänkö näytettävät arvot päivä-, viikko- tai kuukausikohtaisesti, vai käsitelläänkö sitä jatkuvana muuttujana. Tableaussa voi myös piirtää kahden kaaviotyypin, esimerkiksi pylväs- ja viivadiagrammin yhdistelmän, ja muutenkin ohjelmistossa on varsin suuri määrä erilaisia valintoja, joilla voi muokata piirrettävää kaaviota. Siitä huolimatta sillä

ei päästä samanlaiseen muokattavuuteen kuin kaavioiden elementeistä tekemiseen pohjautuvilla kirjastoilla.

Visualisointiin on siis monenlaisia valmiita työkaluja. Suurin ero kehitetyn kirjaston ja valmiiden työkalujen välillä on se, että valmiit työkalut pyrkivät olemaan yleiskäyttöisiä ja helppoja käyttää, kun taas työssä tehty kirjasto on tehty vain projektin tarpeisiin. Siksi valmiit kirjastot eivät välttämättä sovi tiettyyn tarkoitukseen, kuten esimerkiksi tehtävänhallintaohjelmistojen tehtävien visualisointiin yhtä hyvin, kuin tarkoitusta varten tehdyt visualisaatiot. Kirjastoa tehdessä mahdollisesti painotettiin liikaa yleiskäyttöisyyttä, mikä lisäsi tarpeettomasti työmäärää, sillä loppujen lopuksi ainoastaan version- ja tehtävänhallintadatan visualisointi oli tavoitteena. Tämä taas johti siihen, että kirjaston käyttäminen, erityisesti datan jäsentäminen, ei ole automatisoitua, vaan käyttäjä joutuu tekemään sen itse.

7. JOHTOPÄÄTÖKSET

Työssä kehitettiin iteratiivisesti visualisointikirjastoa, jonka tarkoituksena on helpottaa visualisaatioiden tekemistä Need for Speed -projektissa tutkiville tutkijoille. Projekti pyrkii parantamaan suomalaisten ohjelmistoalan yritysten kilpailukykyä, ja ohjelmistoanalytiikka on tähän yksi työkalu. Projektissa visualisoitiin tehtävänhallintaohjelmista saatua dataa yhdessä versionhallintajärjestelmistä saadun datan kanssa. Esimerkiksi visualisoitiin projektien tehtävänhallinnassa olevien tehtävien määrää, sekä niiden elinaikaa. Tällaisista visualisaatioista pystyy jonkin verran tekemään päätelmiä projektin tilasta, esimerkiksi virheisiin liittyvien tehtävien määrän kasvaminen projektin aikana voi kertoa että jotain on vialla. Tapauskohtaisesti kehitettyjen visualisaatioiden uudelleenkäytettävyys oli kuitenkin heikko, ja siksi pyrittiin kehittämään projektin tarpeisiin uudelleenkäytettävä visualisointikirjasto. Kehitys oli iteratiivista, ja projektin aikana kehitettiin useita versioita, joissa pyrittiin parantamaan edellisen version puutteita.

Työn tuloksena saatiin kokeiluna toimiva visualisointikirjasto, mutta tämän version hyödyllisyys verrattuna täysin tarkoitusta varten tehtyihin visualisointeihin jäi rajoitetuksi. Pääasiallisena syynä tähän on monien visualisaation tekemiseen menevien työvaiheiden manuaalisuus. Data tulee ensin käsitellä csv-tiedostoihin, mikä tehtiin ajamalla Python-komentosarjoja käsin. Tämän jälkeen data tulee vielä jäsentää kaavioiden ymmärtämään muotoon, minkä toteuttamiseen Javascriptillä menee helposti useita työtunteja. Työkalua tehdessä kuitenkin opittiin, että tämä lähestymistapa ei toiminut, ja saatujen tulosten perusteella lähdettiin kehittämään seuraavaa versiota, jossa tämän version ongelmiin on kiinnitetty huomiota. Ylipäätään työkalua tehdessä opittiin, että tiimin sisäinen kommunikointi on erittäin tärkeää, ja yleiskäyttöisen työkalun tekeminen ei ole helppoa. Yleiskäyttöistä työkalua tehdessä pitäisi olla varsin tarkkaan selvillä, mitä työkalulta halutaan. Muuten käy helposti niin, että työkalusta tulee liian vaikeakäyttöinen, tai sen kehittämiseen kuluu huomattavasti ylimääräistä vaivaa.

LÄHTEET

[1] Dygraphs. http://dygraphs.com/, Nov 2014.

[2] Hackystat tutorial. https://code.google.com/p/hackystat/wiki/Tutorial_GuidedTour, Nov 2014.

[3] The prefuse visualization toolkit. http://prefuse.org/, Nov 2014.

[4] Simple example of mvc (model view controller) design pattern for abstraction. http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design, Jan 2015.

[5] Fumio Akiyama. An example of software system debugging. In *IFIP Congress (1)'71*, pages 353–359, 1971.

[6] Atlassian. Jira. https://www.atlassian.com/software/jira, Oct 2014.

[7] Atlassian. Jira database schema. https://developer.atlassian.com/display/JIRADEV/Database+Schema, Jan 2015.

[8] Atlassian. Jira demo. https://jira.atlassian.com/, Jan 2015.

[9] Dmitry Baranovskiy. Raphael - javascript library. http://raphaeljs.com/, Nov 2014.

[10] O. Baysal, R. Holmes, and M.W. Godfrey. Developer dashboards: The need for qualitative analytics. *Software, IEEE*, 30(4):46–52, July 2013.

[11] Christian Bird, Nachiappan Nagappan, Premkumar Devanbu, Harald Gall, and Brendan Murphy. Does distributed development affect software quality?: An empirical case study of windows vista. *Commun. ACM*, 52(8):85–93, August 2009.

[12] Christian Bird and Thomas Zimmermann. Assessing the value of branches with what-if analysis. In *Proceedings of the 20th International Symposium on Foundations of Software Engineering (FSE 2012)*. Association for Computing Machinery, Inc., November 2012.

[13] Mike Bostock. D3 data-driven documents. http://d3js.org/, Jun 2014.

[14] Carlos Garcia Campos. Cvsanaly. http://gsyc.es/~carlosgc/files/cvsanaly.pdf, Apr 2009.

[15] Scott Chacon. *Pro Git*. Apress, Berkely, CA, USA, 1st edition, 2009.

[16] J. Czerwonka, R. Das, N. Nagappan, A. Tarvo, and A. Teterev. Crane: Failure prediction, change analysis and test prioritization in practice – experiences from windows. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pages 357–366, March 2011.

[17] J. Czerwonka, N. Nagappan, W. Schulte, and B. Murphy. Codemine: Building a software development data analytics platform at microsoft. *Software, IEEE*, 30(4):64–71, July 2013.

[18] Digile. Need for speed. `http://www.n4s.fi/wordpress/wp-content/uploads/2014/03/SRA-Need4Speed-4-2.pdf`, Jul 2014.

[19] Tore Dyba and T. Dingsoyr. What do we know about agile software development? *Software, IEEE*, 26(5):6–9, Sept 2009.

[20] Google. Google charts. `https://developers.google.com/chart/`, Jul 2014.

[21] Shi Han, Yingnong Dang, Song Ge, Dongmei Zhang, and Tao Xie. Performance debugging in the large via mining millions of stack traces. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 145–155, Piscataway, NJ, USA, 2012. IEEE Press.

[22] Watts S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[23] SAS Institute Inc. Sas: Business analytics and business intelligence software. `http://www.sas.com/en_us/home.html`, Nov 2014.

[24] P.M. Johnson. Leap: a "personal information environment"for software engineers. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 654–657, May 1999.

[25] P.M. Johnson. Searching under the streetlight for useful software analytics. *Software, IEEE*, 30(4):57–63, July 2013.

[26] P.M. Johnson, Hongbing Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, Shenyan Zhen, and W.E.J. Doane. Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 641–646, May 2003.

[27] Akseli Kelloniemi. Cvsanaly asennusohje, 2014.

[28] T. Lehtonen, V.-P. Eloranta, M. Leppanen, and E. Isohanni. Visualizations as a basis for agile software process improvement. In *Software Engineering*

Conference (APSEC, 2013 20th Asia-Pacific, volume 1, pages 495–502, Dec 2013.

[29] Anna-Liisa Mattila, Mikael Niemelä, Kari Systä, Anna Eteläaaho, Outi Sievi-Korte, and Akseli Kelloniemi. What can be revealed by visualizing software engineering data? - a case study. Feb 2015.

[30] T. Menzies and T. Zimmermann. Software analytics: So what? *Software, IEEE*, 30(4):31–37, July 2013.

[31] Massachusetts Institute of Technology and Contributors 2006-2009. Timeline - web widget for visualizing temporal data. `http://www.simile-widgets.org/timeline/`, Nov 2014.

[32] Novus Partners. Nvd3 re-usable charts for d3.js. `http://nvd3.org/`, Oct 2014.

[33] Gregorio Robles, Jesus M. Gonzalez-Barahona, Daniel Izquierdo-Cortazar, Universidad Rey Juan Carlos, Israel Herraiz, and Universidad Alfonso X el Sabio. Tools and datasets for mining libre software repositories. September 2011.

[34] Tableau software. Tableau. `http://www.tableausoftware.com/`, Dec 2014.

[35] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, CT, USA, 1986.

[36] Maurice Wilkes. *Memoirs of a Computer Pioneer.* Massachusetts Institute of Technology, Cambridge, MA, USA, 1985.

[37] Dongmei Zhang, Shi Han, Yingnong Dang, Jian-Guang Lou, Haidong Zhang, and Tao Xie. Software analytics in practice. *Software, IEEE*, 30(5):30–37, Sept 2013.

A. LIITTEITÄ

```
function fileReader() {
    // Expects that file has some sort of similar structure as csv files


   this.loadDataFiles = function(filenames, separators, callback) {
                var returnvalue = []
        this.relayDataWhenReady = function (data) {
                returnvalue.push(data);
                if (returnvalue.length == filenames.length) {
                        callback(returnvalue);
                }
        }
        for (i = 0; i < filenames.length; ++i){
                this.loadDataFile(filenames[i], separators[i], this.relayDataWhenReady);

        }
    }

     this.loadDataFile = function (filename, separator, callback) {
        var dsv = d3.dsv(separator, "text/plain");
        var returnvalue = null;
        dsv(filename, function (error, data) {
            if (error == null) {
                returnvalue = data;
            }
            else {
                console.log("Error when opening file: " + filename);
                return null;
            }
            callback(returnvalue);
        }
        );

       // return returnvalue;
     }

}
```

Listaus 4: Listaus tiedostojen lukijasta.

```
function Parser() {

that = {};

that.parseCommits = function(data) {
   var parseDate = d3.time.format("%Y-%m-%d").parse;
   data.forEach(function(d) {
      d.author_id = +d.author_id;
      d.commits = +d.commits;
      d.loc_added = +d.loc_added;
      d.loc_removed = +d.loc_removed;
      d.time = parseDate(d.time);
   });

   var returnvalue = {groups: [], values: []};
   groupmap = {};
   for (var i = 0; i < data.length; i++) {
      if (!(data[i].author_id in groupmap)) {
         groupmap[data[i].author_id] = true;
         returnvalue.groups.push(data[i].author_id);
      }
      returnvalue.values.push({xvalue: data[i].time, data: data[i].commits, group: data[i].author_id});
   }
   returnvalue.timerange = d3.extent(data, function(d) { return d.time;});

   return returnvalue;
}

that.parseIssues = function (issues, statuschanges) {
   var parseDate = d3.time.format("%Y-%m-%d").parse;
   issues.forEach(function(d) {
      d.created = parseDate(d.created.split("␣")[0]);
      d.duedate = parseDate(d.duedate.split("␣")[0]);
      d.updated = parseDate(d.updated.split("␣")[0]);
      d.resolutiondate = parseDate(d.resolutiondate.split("␣")[0]);
      d.issuenum = +d.issuenum;
      d.id = +d.id;
   });

   issues.sort(function(a, b) {
      return a.id - b.id;
   });

   statuschanges.forEach(function(d) {
      d.id = +d.id;
      d.created = parseDate(d.created.split("␣")[0]);
   });

   statuschanges.sort(function(a, b) {
      if (a.id !== b.id) {return a.id - b.id; }
      else { return a.created - b.created; }
   });


   var temp = new Map;
   var returnvalue = {};
   var types = ["Not␣started", "Open", "In␣Progress", "Closed", "To␣be␣reviewed", "Reopened", "Pending", "Resolved"];
   returnvalue.groups = types;
   for (var i = 0; i < issues.length; i++) {
      issues[i].statuschanges = [];
      temp.put(issues[i].id, issues[i]);
   }

   var issue;
   for (var i = 0; i < statuschanges.length; i++) {
      issue = temp.get(statuschanges[i].id);
      if ( issue !== undefined) {
         issue.statuschanges.push(statuschanges[i]);
      }
   }
   returnvalue.values = [];
   var start;
   for (var i = 0; i < issues.length; i++) {
      if (issues[i].statuschanges.length === 0) {
         returnvalue.values.push({id: issues[i].id, start: issues[i].created, end:
            issues[i].resolutiondate, group: "Not␣started", type: issues[i].pname});
         continue;
      }
      start = issues[i].created;
      for (var j = 0; j < issues[i].statuschanges[j].length; j++) {
```

```
            returnvalue.values.push({id: issues[i].id, start: start, end: issues[i].statuschanges[j].created,
82              group: issues[i].statuschanges[j].oldstring, type: issues[i].pname});
                start = issues[i].statuschanges[j].created;
84          }
            if (issues[i].resolutiondate - start > 0) {
86              returnvalue.values.push({start: start, end: issues[i].resolutiondate, type:
                    issues[i].pname, id: issues[i].id, group: issues[i].statuschanges[j].newstring });
88          }
        }
90
        return returnvalue;
92  }

94  that.parseSprints = function(data, timerange) {
        function sprintTooltip(d) {
96          var timeFormat = d3.time.format("%d.%m.%Y");
            return d.name + "\nStart: " + timeFormat(d.start) + "\nEnd: " + timeFormat(d.end) +
98              "\nComplete: " + timeFormat(d.complete);
        }
100     function noSprintTooltip(d) {
            var timeFormat = d3.time.format("%d.%m.%Y");
102         return "No sprint\nStart: " + timeFormat(new Date(d[0].getTime() + 86400000)) +
                "\nEnd: " + timeFormat(new Date(d[1].getTime() - 86400000));
104     }

106     var parseDate = d3.time.format("%Y-%m-%d").parse;
        data.forEach(function(d) {
108         d.start = parseDate(d.start);
            d.end = parseDate(d.end);
110         d.complete = parseDate(d.complete);
        });
112

114     if (arguments.length == 1) {
            var timerange = {};
116         timerange.start = d3.extent(data, function(d) {return d.start;})[0];
            timerange.end = d3.extent(data, function(d) {return d.end;})[1];
118     }

120     returnvalue = {groups: [0, 1, 2], values: []};
        returnvalue.timerange = timerange;
122
        data.sort(function(a,b) {
124         return a.start - b.start;
        });
126
        for (var i = 0; i < data.length; i++) {
128         returnvalue.values.push({start: data[i].start, end: data[i].end, opacity: (i % 2) / 2,
                group: i % 2, tooltip: sprintTooltip(data[i])});
130     }

132     previous_end = timerange.start;
        for (var i = 0; i < data.length; i++) {
134         if (data[i].start - previous_end > 86400000) {
                returnvalue.values.push({start: previous_end, end: data[i].start, opacity: 0.5,
136                 group: 2, tooltip: noSprintTooltip([previous_end, data[i].start])});
            }
138         previous_end = data[i].end;
        }
140
        if (timerange.end - previous_end > 86400000) {
142         returnvalue.values.push({start: previous_end, end: timerange.end, opacity: 0.5, group: 2,
                tooltip: noSprintTooltip([previous_end, timerange.end])});
144     }

146     returnvalue.values.sort(function(a,b) {
            return a.start - b.start;
148     });

150     return returnvalue;
    }
152
    that.parseLocs = function (data) {
154 }
    return that;
156 }
```

Listaus 5: Listaus jäsentäjästä.

```
function Chart () {

var that = {};

//initialisoidaan kuvaaja diviin, zoomHandler on funktio jolle välitetään
//zoomauksen parametrit, 3 parametri zoomHandler, 4 id ja 5 marginaali
that.initChart = function (div, size, zoomHandler, id, margin) {
    that.svg = div.append("svg")
        .attr("width", size.width)
        .attr("x", 0)
        .attr("y", 0)
        .attr("height", size.height)
        .attr("id", id);

    that.size = {width: size.width, height: size.height};
    that.independent_chart = true;
    that.zoomHandler = zoomHandler;
    that.id = id;
    that.margin = margin;
    that.independent_chart = true;
    that.zoomVar = d3.behavior.zoom().scaleExtent([1, 8]).on("zoom", zoomHandler);
}

that.zoomVar;
var width;
var height;

that.x;
that.y;
that.color;
that.xAxis;
that.yAxis;
that.independent_chart;
that.svg_inner = 0;


//initialisoidaan kuvaaja toisen sisään
that.initInExistingChart = function (chart, zoomHandler, id) {
    that.size = chart.size;
    that.svg = chart.getSvg();
    that.svg_inner = chart.getSvg_inner();
    that.zoomHandler = zoomHandler;
    that.margin = chart.margin;
    that.id = id;
    that.independent_chart = false;
    that.other = chart;
    that.zoomVar = d3.behavior.zoom().scaleExtent([1, 8]).on("zoom", zoomHandler);
}

that.drawChart = function () {
};

that.getSvg = function () {
    return that.svg;
};

that.setColor = function (colors) {
    that.color = colors;
};

that.getSvg_inner = function() {
    return that.svg_inner;
};

that.getSize = function() {
    return that.size;
};

that.getMargin = function() {
    return that.margin;
};

that.clear = function () {
    $("#" + that.id).empty();
};

//zoomausfunktio, tätä pitäisi kutsua muualta
that.zoom = function (t_x, t_y, s) {
    t_x = Math.max(that.width * (1 - s), Math.min(0, t_x));
    t_y = Math.max(that.height * (1 - s), Math.min(0, t_y));
```

```
        that.zoomVar.translate([t_x, t_y]);
82      that.zoomVar.scale(s);

84      that.svg_inner.selectAll("." + that.id)
            .attr("transform", "translate(" + t_x + ",0)scale(" + s + ",␣1)");
86  };

88  return that;
    }
```

Listaus 6: Listaus chartista.

```
function timeLine() {

var chart = Chart();

chart.setColor(["#ffffff", "#9ecae1", "#d9d9d9"]);

chart.drawChart = function(data) {

    //inspects if a tooltip is defined and if not, generates a default tooltip
    function tooltip(d) {
        if (typeof d.tooltip !== 'undefined') {
            return d.tooltip;
        }
        var timeFormat = d3.time.format("%d.%m.%Y");
        return "Start: " + timeFormat(d.start) + "\nEnd: " + timeFormat(d.end);
    }

    //width and height are size without marginals, chart.size with them
    chart.width = chart.size.width - chart.margin.left - chart.margin.right;
    chart.height = chart.size.height - chart.margin.top - chart.margin.bottom;

    var x = d3.time.scale()
        .range([0, chart.width]);

    var y = d3.scale.linear()
        .range([chart.height, 0]);

    //set domains for axis
    y.domain([data.minvalue, data.maxvalue]);
    x.domain([data.timerange.start, data.timerange.end]);

    var colormap = {};

    for (var i = 0; i < data.groups.length; i++) {
        colormap[data.groups[i]] = chart.color[i];
    }

    //If independent chart, generate inner svg element.
    if (chart.independent_chart) {
        $("#" + chart.id).empty();
        chart.svg_inner = chart.svg.append("svg")
            .attr("width", chart.width)
            .attr("x", chart.margin.left)
            .attr("y", chart.margin.top)
            .attr("height", chart.height)
            .call(chart.zoomVar);

        chart.svg_inner.append("rect")
            .attr("width", chart.width)
            .attr("height", chart.height)
            .attr("x", 0)
            .attr("y", 0)
            .style("opacity", 0);
    }
    else {
        chart.svg_inner = chart.other.getSvg_inner();
    }

    //draw timeline, color according to groups
    chart.svg_inner.selectAll("rect").data(data.values)
      .enter().append("rect")
        .attr("x", function(d) { return x(d.start); })
        .attr("y", 0)
        .attr("height", chart.height)
        .attr("width", function(d) {return x(d.end) - x(d.start); })
        .attr("class", chart.id)
        .style("fill", function(d) { return colormap[d.group]; })
        .style("opacity", function(d) { if (typeof d.opacity === "undefined") { return 1;} return d.opacity; })
        .append("svg:title").text(function(d) { return tooltip(d); });

};

return chart;
}
```

Listaus 7: Listaus timelinestä.

```
function lineChart () {

var chart = new Chart();

chart.setColor(["#a6cee3","#1f78b4","#b2df8a","#33a02c","#fb9a99","#e31a1c","#fdbf6f","#ff7f00",
   "#cab2d6","#6a3d9a","#ffff33","#b15928"]);

chart.drawChart = function(data) {

   //width and height are size without marginals, chart.size with them
   chart.width = chart.size.width - chart.margin.left - chart.margin.right;
   chart.height = chart.size.height - chart.margin.top - chart.margin.bottom;

   var x = d3.time.scale()
      .range([0, chart.width]);

   var y = d3.scale.linear()
      .range([chart.height, 0]);

   //set axis domains
   y.domain([data.minvalue, data.maxvalue]);
   x.domain([data.timerange.start, data.timerange.end]);

   //generate map for colors
   var colormap = {};

   for (var i = 0; i < data.groups.length; i++) {
      colormap[data.groups[i]] = chart.color[i];
   }

   //If independent chart, generate inner svg element.
   if (chart.independent_chart) {
      $("#" + chart.id).empty();
      chart.svg_inner = chart.svg.append("svg")
         .attr("width", chart.width)
         .attr("x", chart.margin.left)
         .attr("y", chart.margin.top)
         .attr("height", chart.height)
         .call(chart.zoomVar);

      chart.svg_inner.append("rect")
         .attr("width", chart.width)
         .attr("height", chart.height)
         .attr("x", 0)
         .attr("y", 0)
         .style("opacity", 0);
   }
   else {
      chart.svg_inner = chart.other.getSvg_inner();
   }

       var yAxis = d3.svg.axis()
                .scale(y)
                .orient("right");

   //add y-axis to outer svg element
   chart.svg.append("g")
      .attr("class", "y␣axis")
      .call(yAxis)
      .attr("transform", "translate("  + (chart.margin.left + chart.width) + ",␣" + chart.margin.top + ")")
     .append("text")
      .attr("transform", "rotate(-90)")
      .attr("y", -6)
      .attr("dy", ".71em")
      .text("LOC");

   //define line for
   var line = d3.svg.line()
      .x(function(d) { return x(d.time); })
      .y(function(d) { return y(d.data); });

   var linedata = [];

   //turn data into suitable form for drawing as a line, here
   //an object is generated for each line
   for (var i = 0; i < data.groups.length; i++) {
      linedata.push({name: data.groups[i], color: chart.color[i], values: []});
   }

   //add data to values table in each line
```

```
    for (var i = 0; i < data.values.length; i++) {
        linedata[data.groups.indexOf(data.values[i].group)].values.push({time: data.values[i].time,
            data: data.values[i].data});
    }

    lines = chart.svg_inner.selectAll(".locs")
        .data(linedata)
      .enter().append("g")
        .attr("class", chart.id);

    //make line for each dataset
    lines.append("path")
        .attr("class", "line")
        .attr("d", function(d) { return line(d.values); })
        .style("stroke", function(d) { return d.color; })
        .style("fill", "none")
        .style("stroke-width", "2.5");

    //generate legend
    var legend = chart.svg.selectAll(".legend")
        .data(data.groups)
      .enter().append("g")
        .attr("class", chart.id)
        .attr("transform", function(d, i) { return "translate(0," + i * 20 + ")"; });

    legend.append("rect")
        .attr("x", chart.size.width - 100)
        .attr("width", 18)
        .attr("height", 18)
        .style("fill", function(d) {return colormap[d]; });


    legend.append("text")
        .attr("x", chart.size.width - 100)
        .attr("y", 9)
        .attr("dy", ".35em")
        .style("text-anchor", "end")
        .text(function(d) { return d; });
};

return chart;
}
```

Listaus 8: Listaus linechartista.

```
function horBarChart () {

var chart = Chart();

chart.setColor(["#a6cee3","#1f78b4","#b2df8a","#33a02c","#fb9a99","#e31a1c","#fdbf6f","#ff7f00",
   "#cab2d6","#6a3d9a","#ffff33","#b15928"]);

chart.drawChart = function(data) {

   //initialization
   var zoom = d3.behavior.zoom().scaleExtent([1, 8]).on("zoom", chart.zoomHandler);
   var barwidth;
   chart.width = chart.size.width - chart.margin.left - chart.margin.right;
   chart.height = chart.size.height - chart.margin.top - chart.margin.bottom;
   barwidth = Math.max(Math.floor(chart.height / data.values.length - 1), 1);

   //generate map for colors
   var colormap = {};
   for (var i = 0; i < data.groups.length; i++) {
      colormap[data.groups[i]] = chart.color[i];
   }

   var x = d3.time.scale()
      .range([0, chart.width]);

   x.domain([data.timerange.start, data.timerange.end]);

   //If independent chart, generate inner svg element.
   if (chart.independent_chart) {
      $("#" + chart.id).empty();

      chart.size.height = chart.height + chart.margin.top + chart.margin.bottom;
      chart.svg.attr("height", chart.size.height);
      chart.svg_inner = chart.svg.append("svg")
         .attr("width", chart.width)
         .attr("x", chart.margin.left)
         .attr("y", chart.margin.top)
         .attr("height", chart.height)
         .call(zoom);

      chart.svg_inner.append("rect")
         .attr("width", chart.width)
         .attr("height", chart.height)
         .attr("x", 0)
         .attr("y", 0)
         .style("opacity", 0);
   }
   else {
      chart.svg_inner = chart.other.getSvg_inner();
   }

   var y = d3.scale.ordinal()
      .rangePoints([0, chart.height - barwidth]);

   //set domain for y, in this case they are ids in order
   y.domain(data.values.map(function(d) { return d.id; }));

   //select all rectangles
   var rect = chart.svg_inner.selectAll("rect." + chart.id);

   //give data, and generate rectangles depending on the time range. y-axis according to id
   rect.data(data.values)
     .enter().append("rect")
      .attr("x", function(d) { return x(d.start); })
      .attr("y", function(d) { return y(d.id); })
      .attr("class", chart.id)
      .attr("height", barwidth)
      .attr("width", function(d) { if (d.end === null) { return chart.width - x(d.start); }
                                   else { return x(d.end) - x(d.start); } })
      .attr("class", "issues")
      .style("fill", function(d) { return colormap[d.group]; })
      .append("svg:title").text(function(d) { return tooltip(d); });

   //select legend
   var legend = chart.svg.selectAll(".legend")
      .data(data.groups)
     .enter().append("g")
      .attr("class", "legend")
      .attr("transform", function(d, i) { return "translate(0," + i * 20 + ")"; });

```

```
        legend.append("rect")
82          .attr("x", chart.size.width - 100)
            .attr("width", 18)
84          .attr("height", 18)
            .style("fill", function(d) {return colormap[d]; });
86
        legend.append("text")
88          .attr("x", chart.size.width - 100)
            .attr("y", 9)
90          .attr("dy", ".35em")
            .style("text-anchor", "end")
92          .text(function(d) { return d; });

94      //generoidaan oletustooltippi jos sitä ei ole määritelty
        function tooltip(d) {
96          var timeFormat = d3.time.format("%d.%m.%Y");
            var endDate;
98          if (typeof d.tooltip !== 'undefined') {
                return d.tooltip;
100         }
            else if (d.end === null) {
102             endDate = d.end;
            }
104         else {
                endDate = timeFormat(d.end);
106         }
            return "ID: " + d.id + "\nGroup: " + d.group + "\nStatus start: " + timeFormat(d.start) +
108             "\nStatus end: " + endDate;
        }
110
    };
112
    return chart;
114 }
```

Listaus 9: Listaus horbarchartista.

```
function stackedVertBarChart() {

var chart = Chart();

chart.setColor(["#a6cee3","#1f78b4","#b2df8a","#33a02c","#fb9a99","#e31a1c",
                "#fdbf6f","#ff7f00","#cab2d6","#6a3d9a","#ffff33","#b15928",
                                "#FFC574", "#ad942e", "#c9ba7c", "#d3ddea", "#c417bb",
                                "#ddb8db", "#4f701c", "#ffd800", "#965851", "#116675", "#13ad53",
                                "#6edd9d", "#51123e", "#ceb9c8", "#20773e", "#7a9b86", "#bababa",
                                "#8941d4","#97e241", "#753d23","#e8dda2", "#6d840f", "#686157",
                                "#7c4e89","#031a4c", "#a0a0a0", "#1ea082"]);

chart.setAxisType = function(type) {
   chart.xType = type;
};

chart.showXAxis = function(show) {
   chart.showX = show;
};
chart.showX = true;

//reimplement zooming to zoom x axis
chart.zoom = function (t_x, t_y, s) {
   t_x = Math.max(chart.width * (1 - s), Math.min(0, t_x));
   t_y = Math.max(chart.height * (1 - s), Math.min(0, t_y));
   chart.zoomVar.translate([t_x, t_y]);
   chart.zoomVar.scale(s);

   chart.svg.select(".x_" + chart.id + ".axis").call(chart.xAxis);
   chart.svg_inner.selectAll("." + chart.id)
      .attr("transform", "translate(" + t_x + ",0)scale(" + s + ",␣1)");
};

chart.drawChart = function(data) {

   //inspects if a tooltip is defined and if not, generates a default tooltip
   function columnTooltip (d) {
      if (typeof d.tooltip !== 'undefined') {
         return d.tooltip;
      }
      else if (chart.xType === 'time') {
         var timeFormat = d3.time.format("%d.%m.%Y");
         return "Date:␣" + timeFormat(d.xvalue) + "\nGroup:␣" + data.groups[d.group - 1] + "\nData:␣" + d.data;
      }
      else {
         return "ID:␣" + d.xvalue + "\nGroup:␣" + data.groups[d.group - 1] + "\nData:␣" + d.data;
      }
   }

   //sorts data into order, groupmap is made to avoid linear search when looking
   //for index of group
   data.values.sort(function(a, b) {
      if (a.xvalue === b.xvalue) {
         return data.groups.indexOf(a.group) - data.groups.indexOf(b.group);
      }
      else {
         return a.xvalue - b.xvalue;
      }
   });

   //if timerange.start and timerange.end are not defined, use range in data
   if (typeof data.timerange === "undefined") {
      data.timerange = {};
   }
   if (typeof data.timerange.start === "undefined") {
      data.timerange.start = data.values[0].xvalue;
   }
   if (typeof data.timerange.end === "undefined") {
      data.timerange.end = data.values[data.values.length - 1].xvalue;
   }

   //generate map for colors
   var colormap = {};

   for (var i = 0; i < data.groups.length; i++) {
      colormap[data.groups[i]] = chart.color[i];
   }

   //width and height are size without marginals, chart.size with them
   chart.width = chart.size.width - chart.margin.left - chart.margin.right;
```

```
    chart.height = chart.size.height - chart.margin.top - chart.margin.bottom;
82  var barwidth;

84  //Setting x scale type and range. It's either
    //time or ordinal scale.
86  if (chart.xType !== "ordinal") {
        chart.x = d3.time.scale()
88          .range([0, chart.width]);
        chart.x.domain([data.timerange.start, data.timerange.end]);
90      xstart = data.timerange.start;
            chart.xAxis = d3.svg.axis()
92        .scale(chart.x)
          .orient("bottom");
94
        barwidth = Math.floor(chart.width / ((data.timerange.end.getTime() -
96          data.timerange.start.getTime()) / 86400000) - 1)
    }
98  else {
        ids = [];
100     used_ids = {};
        for (var i = 0; i < data.values.length; i++) {
102         if (!(data.values[i].xvalue in used_ids)) {
                used_ids[data.values[i].xvalue] = true;
104             ids.push(data.values[i].xvalue);
            }
106     }
        chart.x = d3.scale.ordinal()
108         .domain(ids)
            .rangeRoundBands([0, chart.width], .1);
110     chart.xAxis = d3.svg.axis()
          .scale(chart.x)
112       .orient("bottom");

114     chart.xAxis.tickValues(x.domain().filter(function(d, i) { return !(i % Math.floor(
            Math.max(ids.length, 10) / 10)); }));
116     barwidth = Math.max(Math.floor(chart.width / Math.max((ids.length - 1), 1) - 1), 1);

118 }

120 chart.y = d3.scale.linear()
        .range([chart.height, 0]);
122
    var columns = [];
124 //hirveä purkka, jostain syystä d3 ohittaa ekat n arvoa, joten generoidaan dummyarvoja.
    for (var i = 0; i < 50; i++) {
126     columns.push({xvalue: data.timerange.start, group: data.groups[0], end: 0,
            start: 0, tooltip: "", color: "#ffffff"});
128 }
    var xstart;
130 var end = 0;
    var start = 0;
132
    //Data is changed into form from which it can be drawn. As it's a stacked column chart, data needs
134 //to be changed into a form where there is a cumulative start and end value between which the bar
    //will be drawn.
136
    for (var i = 0; i < data.values.length; i++) {
138     if (data.values[i].xvalue !== xstart) {
            xstart = data.values[i].xvalue;
140         start = 0;
            end = data.values[i].data;
142         columns.push({xvalue: data.values[i].xvalue, group: data.values[i].group, start: start, end: end,
                tooltip: columnTooltip(data.values[i]), color: colormap[data.values[i].group]});
144         start = end;
        }
146     else {
            end = end + data.values[i].data;
148         columns.push({xvalue: data.values[i].xvalue, group: data.values[i].group, start: start, end: end,
                tooltip: columnTooltip(data.values[i]), color: colormap[data.values[i].group]});
150         start = end;
        }
152 }

154       chart.yAxis = d3.svg.axis()
                .scale(chart.y)
156             .orient("left");

158 //if minvalue or maxvalue aren't defined, use values in data
    if (typeof data.minvalue === "undefined") {
160     data.minvalue = 0;
```

```
    }
    if (typeof data.maxvalue === "undefined") {
        data.maxvalue = d3.extent(columns, function(d) { return d.end;})[1];
    }

    //set y-axis value range
    chart.y.domain([data.minvalue, data.maxvalue]);

    //redefine zooming to use reimplemented zoom function
    chart.zoomVar = d3.behavior.zoom().x(chart.x).scaleExtent([1, 8]).on("zoom", chart.zoomHandler);

    //If independent chart, generate inner svg element.
    if (chart.independent_chart) {
        $("#" + chart.id).empty();
        chart.svg_inner = chart.svg.append("svg")
            .attr("width", chart.width)
            .attr("x", chart.margin.left)
            .attr("y", chart.margin.top)
            .attr("height", chart.height)
            .call(chart.zoomVar);

        chart.svg_inner.append("rect")
            .attr("width", chart.width)
            .attr("height", chart.height)
            .attr("x", 0)
            .attr("y", 0)
            .style("opacity", 0);
    }
    else {
        chart.svg_inner = chart.other.getSvg_inner();
    }
    //add title
    chart.svg.append("text")
        .attr("x", chart.size.width / 2)
        .attr("y", chart.margin.top / 2)
        .attr("text-anchor", "middle")
        .style("font-size", "16px");

    //add y-axis
    chart.svg.append("g")
        .attr("class", "y_" + chart.id + "␣axis")
        .call(chart.yAxis)
        .attr("transform", "translate(" + chart.margin.left + ",␣" + chart.margin.top + ")")
      .append("text")
        .attr("transform", "rotate(-90)")
        .attr("y", -6)
        .attr("dy", ".71em");

    //add x-axis if it is to be added
    if (chart.showX) {
        chart.svg.append("g")
            .attr("class", "x_" + chart.id + "␣axis")
            .call(chart.xAxis)
            .attr("transform", "translate(␣" + chart.margin.left + "," + (chart.margin.top + chart.height) + ")")
          .append("text")
            .attr("transform", "rotate(-90)")
            .attr("y", -6)
            .attr("dy", ".71em");
    }

    //select rectangles from inner svg element
    var rect = chart.svg_inner.selectAll("rect");

    //set data and values for the rectangles
    rect.data(columns)
      .enter().append("rect")
        .attr("x", function(d) { return chart.x(d.xvalue); })
        .attr("y", function(d) { return chart.y(d.end); })
        .attr("height", function(d) { return chart.y(d.start) - chart.y(d.end); })
        .attr("width", barwidth)
        .attr("class", chart.id)
        .style("fill", function(d) { return d.color; })
        .append("svg:title").text(function(d) {return d.tooltip; });


    //make a selection for legend
    var legend = chart.svg.selectAll(".legend")
        .data(data.groups)
      .enter().append("g")
        .attr("class", chart.id)
```

```
        .attr("transform", function(d, i) { return "translate(0," + i * 20 + ")"; });

    //add rectangle for indicating color
    legend.append("rect")
        .attr("x", chart.size.width - 100)
        .attr("width", 18)
        .attr("height", 18)
        .style("fill", function(d) {return colormap[d]; });

    //name of group
    legend.append("text")
        .attr("x", chart.size.width -100)
        .attr("y", 9)
        .attr("dy", ".35em")
        .style("text-anchor", "end")
        .text(function(d) { return d; });
};

return chart;
}
```

Listaus 10: Listaus stackedvertbarchartista.